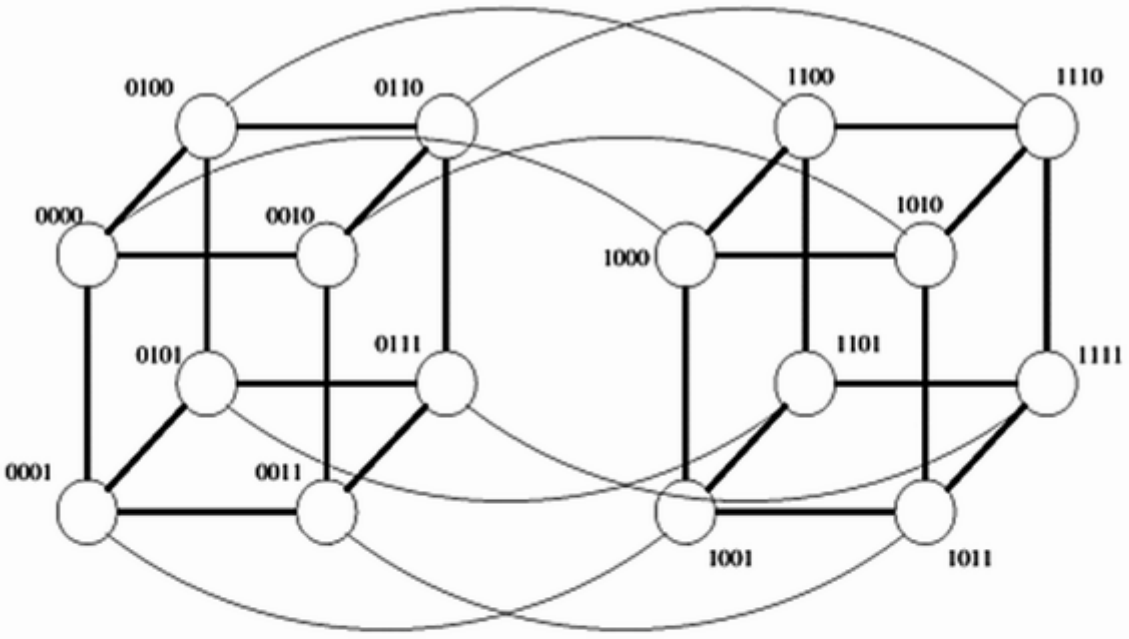
المملكة العربية السعودية
وزارة التعليم العالي
جامعة الملك سعود-فرع القصيم
كلية العلوم-قسم الحاسب الآلي

الحاسبات المتوازية

والخوارزميات المتوازية «دراسة»



إعداد

محمّد عبدالله الجارالله
بسّام عبدالرحمن الخريف
نواف مقبل الحربي
عمر صالح البهدل

إشراف

د. عماد فتاش

المحتويات

المقدمة

الحمد لله رب العالمين، والصلاة والسلام على سيد المرسلين، نبينا محمد، وعلى آله وصحبه أجمعين... أما بعد:

فمنذُ أن بزغ علم الحاسبات الآلية إلى الوجود والعلماء يبذلون جهدهم سعياً لجعل الحاسبات تحل المسائل بشكل أفضل و أسرع، وقد أثمرت التقنية تحسناً في الدوائر الكهربائية، وأصبح بالإمكان وضع العديد منها على شريحة واحدة، كذلك ازدادت سرعة نبضة الساعة للجهاز مما أدى إلى وصول سرعة المعالجات إلى حدود سرعات عالية تقاس بالجيجا هيرتز!. ومع ذلك فهناك قيود طبيعية تتحكم بالمدى الذي يمكن فيه تحسين الأداء لمعالج واحد، فالحرارة مثلاً أو التشويش الكهرومغناطيسي تقللان من كثافة الترانزستورات على الشريحة، وحتى لو توصل الصناع لحل هذه المشاكل فإن سرعة المعالج لا يمكن أبداً أن تتجاوز سرعة الضوء. وعلاوة على هذه القيود الطبيعية فثمّة قيود اقتصادية، ففي وقت ما ستتزايد كلفة إنتاج المعالج السريع جداً بشكل كبير مما قد يؤدي إلى عدم الرغبة بتحمل هذه الكلفة الزائدة. كل هذه الأسباب التي ذكرناها ستؤدي في النهاية إلى ترك جميع الطرق الغير مجدية و تركيز الاهتمام على طريقة واحدة وهي توزيع حمل أداء العمليات الحسابية بين عدة معالجات أو ما يعرف بـ " التوازي ".

وللدلالة على أهمية التوازي المتزايدة يوماً بعد يوم فالحاسبات الشخصية الحديثة بدأت مؤخراً بالاستفادة من التوازي بشكل عملي، فمثلاً يمكن في الوقت الحالي لأي شخص أن

يمتلك حاسباً شخصياً ذا معالجين يعملان بالتوازي؛ فشركة أنتل قد طورت لوحات رئيسية يمكن أن يوضع عليها معالجان من المعالجات الحديثة.

إنه على أهمية موضوع التوازي وشدة الحاجة إليه في الوقت الراهن إلا أنك لا تكاد تجد مراجع عربية تتحدث عن الموضوع أو تعطي لمحة عنه؛ الأمر الذي دفعنا للاعتماد على مراجع أجنبية، مما سبب لنا صعوبات بالغة من ناحية تعريب المصطلحات المتعلقة بالموضوع.

يتكون العمل من مقدمة وأربعة فصول وثبت بالمراجع المستخدمة في العمل، احتـوى **الفصل الأول** فيه على عدد من المفاهيم الأساسية الخاصة بالتوازي و تمت دراسة الأشـكال المختلفة لمعالجة المعطيات على التوازي بالإضافة إلى إيجاز لتاريخ الحاسـبات بـشكل عـام وظهور التوازي خلال هذه الفترات التاريخية.

وقد تضمن **الفصل الثاني** بشكل واسع تصنيف فلايـن[Flynn] للحاسـبات المتوازيـة بالإضافة إلى شبكات الربط وأنواعها المستخدمة في الحاسبات المتوازية ومبدأ عملها.

أما **الفصل الثالث** فقد كُرّس لدراسة الخوارزميات المتوازية و التقنيـات المـستخدمة في كتابة الخوارزميات وقد شُرحت العديد من الأمثلة المباشرة والشهيرة في الحالة التسلـسلية و كيفية استخدام تلك التقنيات لجعلها خوارزميات متوازية.

وعالج **الفصل الرابع** مسألة البرمجة المتوازية و مميزاتها وقد ناقش على وجه التحديد البرمجة بلغة OCCAM المتوازية، وكذلك البرمجة المتوازية بلغة FORTRAN-90، وقد تم بتوسـع دراسة البرمجة المتوازية من خلال لغة C++ باستخدام مكتبة MPI وكيفية كتابـة برنـامج متوازي بهذه اللغة وتم القيام بكتابة عدة برامج تطبيقية متوازية باستخدام لغة C++ و مكتبة MPI لعدد من المسائل المعروفة في الحالة التسلسلية.

هدف المشروع

إن الهدف من هذا المشروع (الحاسبات المتوازية والخوارزميات المتوازية) هو التعريـف بمفهوم التوازي والحاسبات المتوازية من حيث النشأة ومبدإ العمل، بالإضافة إلى التعرّف على طبيعة الخوارزميات المستخدمة لحل المسائل المتعلقة باستخدام الحاسب المتوازي مقارنة مـع التسلسلي، وكتابة برامج متوازية لعدد من المسائل المعروفة.

ونأمل من الله -تعالى- أن نكون قد وفقنا في **مسعانا.**

الطلاب:

محمد بن عبدالله الجارالله
نوّاف بن مقبل الحربي
بسّام بن عبدالرحمن الخريف
عمر بن صالح البهَدل

٢١-ذو القعدة-١٤٢٤ هـ

Parallel_computers@yahoo.com

الفصل الأول: التوازي

لعدة سنوات خلت، كانت الحاسبات المتوازية لا توجد إلا في معامل الأبحاث. أما اليوم فهذه الحاسبات متوفرة وعلى نطاق واسع في المجالات التجارية. إن مجال المعالجة المتوازية قد نضج إلى الحد الذي أصبح يدرس في المراحل الجامعية الأولى. وتجدر الإشارة إلى أن التوازي يغطي طيفاً واسعاً من الأشياء بدايةً من تصميم أبسط مكونات العتاد كالجامع(adder) مثلاً وحتى تحليل النماذج النظرية للحساب المتوازي. وفي الحقيقة فإن جوانب المعالجة المتوازية يمكن أن يتم دمجها في أي مقرر لعلوم الحاسب، مثل دراسة عمارة الحاسبات، أو البرمجة، أو الشبكات أو الخوارزميات وغيرها. وفي هذا الفصل عرض لأهم مفاهيم ومبادئ المعالجة المتوازية.

1.1 مفهوم التوازي

عندما يتحدث خبراء الحاسب عن المعالجة المتوازية، فإنهم لا يعنون معالجة المقولة "الخطان المتوازيان اللذان لا يلتقيان" و لكنهم يناقشون عدداً من الأنشطة الحسابية التي تحدث في وقت واحد.

التوازي parallelism: هو مجموعة من الأنشطة التي تحدث في نفس الوقت.

إن مفهوم التوازي بشكل عام ليس حكراً على علم الحساب، بل ربما أننا نمارس التوازي في حياتنا اليومية، فعلى سبيل المثال كتابة الملاحظات أثناء الاستماع إلى محاضرة تعتبر أنشطة متوازية.

والحساب المتوازي يبدو وكأنه تصور معقول. فالعديد من المفاهيم الموجودة في الحساب المتوازي لها ما يماثلها من المفاهيم في الحياة الاجتماعية، مثل إدارة الأعمال. ففكرة أن العديد من الناس يعملون سوياً لتحقيق هدف واحد هي مشابة لعدد من المعالجات تعمل سوياً نحو هدفٍ واحد. وفكرة الحاجة إلى تقسيم العمل لكي تعمل جميع المعالجات ولا يكون هناك أحد متوقف عن العمل هي فكرة شبيهة للحاجة إلى جعل فريق يعمل دون أن ينتظر أحد الحصول على معلومات من شخص آخر.

ومن هذا التمثيل نرى كيف أن الحساب المتوازي هو نتيجة طبيعية لمفهوم فَرِّقْ-تَسُدْ(divide-and-conquer) . فنبدأ أولاً مع المسألة التي نريد حلها، ثم نحصل على الموارد المتاحة التي يمكن أن نستخدمها لحل المسألة (وفي حالة الحوسبة ستكون هذه الموارد عبارة عن عدد من المعالجات)، بعد ذلك نحاول تقسيم المسألة إلى أجزاء أصغر يمكن أن تؤدى بشكل متزامن بواسطة عدة أشخاص في الفريق.

ومع ذلك يجب أن نكون على حذر لأن العبارة "في نفس الوقت" غير دقيقة، على سبيل المثال الحاسبات السريعة ، مثل الحاسب VAX، قد يبدو وكأننا تقوم بعمل حسابات للعديد من المستخدمين في نفس الوقت وذلك لأن المعالج ينقل المعلومات بسرعة كبيرة. وهنا يصبح لدينا "وهم" أو "انخداع" بأن التعليمات تنفذ بشكل آني وهو ما يطلق عليه "التوازي الوهمي". وبما أن المعالج يقوم بتنفيذ التعليمات لمهمة واحدة فإنه ليس تواز حقيقي.

ويجب التفريق بين مفهوم التزامن(**Concurrency**) ومفهوم التوازي الذي عرضناه آنفاً

التزامن Concurrency: هو القدرة على التشغيل في نفس الوقت.

إن هذا يُظهرُ وكأن التزامن و التوازي لفظان مترادفان لمعنى واحد. ولكن هناك فـرق خفي بينهما. فنحن نستخدم التوازي للإشارة إلى الحالات التي تحدث فيها العمليات في نفس الوقت، مثل أربع مهام يتم تنفيذها على أربع معالجات (CPUs) في نفس الوقت. أما التزامن فيشير إلى كلا الحالتين الأولى التي تعتبر متوازية حقاً والأخرى التي فيها "تـوازي وهمـي" ، وكمثال للتزامن: أربع مهام يقسم بينها الوقت وتنفذ على معالج واحد.

1.2 الحاجة إلى استخدام التوازي

إنه من المفيد الإجابة على السؤال الملح والهام : لماذا نستخدم التوازي؟

إنّ السبب الرئيسي لاستعمال التوازي في تصميم(البرمجيات أو العتاد) هو مـن أجـل الحصول على الأداء الأعلى أو السرعة العالية. وكل أنـواع الحاسـبات الـضخمة اليـوم (Supercomputers) تستخدم التوازي على نطاق واسع لزيادة الأداء، فأسرع حاسـب في العالم في عام 2003 هو الحاسب الياباني "محاكي الأرض" الذي يَستَخدِم أكثر مـن خمـسة آلاف معالج تعمل بالتوازي. ولقد زادت سرعة الحاسبات إلى الحد الذي وصلت فيه الدارات الحاسوبية حدوداً فيزيائية مثل سرعة الضوء. لذلك فمن أجل تحـسين الأداء فـلا بـد أن نستخدم التوازي.

إن السرعة ليست هي السبب الوحيد في استعمال التوازي. فمصمم الحاسب يمكنـه أن يضاعف من المكونات ليزيد من إمكانية الاعتماد على الجهاز(الوثوقية). على سـبيل المثـال نظام توجيه مركبات الفضاء يتكون من ثلاثة أجهزة من الحواسب والتي تقارن نتائج كـل منهم مع الآخر. ويمكن للمركبة أن تسير بجهاز واحد فقط بينما الجهازين الآخرين يكونـان في وضع احتياطي.

ويمكن كذلك استخدام التوازي لجعل السيطرة لا مركزية. فالبنـك مـثلاً يمكنـه أن يستخدم شبكة من الحاسبات الصغيرة في المقر الرئيسي والفروع بدلا من استخدام حاسـب واحد كبير، هذه المعالجة التقسيمية للحاسبات تتميز بوجود التحكم الداخلي عـن طريـق مديري البنك.

يعتبر التوازي نموذجاً مهماً لحل المسائل، فالطبيعة متوازية فبينما تتحدث مع أصدقائك، فإن القلب يضخ الدم، والرئتين تتنفس الهواء، والعينان تتحرك، واللسان يتحرك كل ذلـك يحدث بالتوازي. والكثير من الأشياء متوازية بطبيعتها. مثلاً لاحظ أفعال حشد من النـاس ينتظرون المصعد للصعود إلى أعلى(أنشطة)، وأفراد من الخارج يقومون بـضغط الـزر في الطابق الذي هم متواجدون فيه (حدث) في نفس الوقت الذي يقوم أفراد من داخل المصعد بالضغط على الزر، ولكي نجعل الأداء أمثل (باستخدام برنامج حاسوبي مثلاً) يجب التعامـل مع هذه الأنشطة والأحداث المتوازية.

1.3 فوائد تعدد المعالجات

لتعدد المعالجات فوائد عدة منها:

- تنفيذ المهام المستقلة بمعالجات متزامنة وبذلك تزداد نسبة العمل وعدد المستخدمين.
- الإقلال من تكامل المعالجات في نظام وحيد الكلفة المادية لاشتراكه في نظام لموارد مثل الذاكرة والأقراص ووحدات الربط مع الشبكات.
- يقدم سرعة عالية في التوصيل بين المعالجات المتعددة وينفذ أفضل تناسق وأسرع وصل بين المهام المرتبطة .

• يجزئُ العمل الوحيد الكبير إلى عدة مهام متشابة تنفذ في وقت واحد من أجل السرعة في التطبيق .

1.4 دراسة المعالجة المتوازية

في السنوات الأخيرة أحدثت المعالجة المتوازية ثورة علمية كـبيرة في مجـال الحاسـبات وبدأت تدخل العالم كل يوم عن طريق معالجة البيانات في شكل قواعد بيانـات موزعـة. والمبرمجون العلميون بحاجة لأن يفهموا مبادئ المعالجة المتوازية ليقومـوا ببرمجـة حاسـبات المستقبل.

وجميع الحاسبات العملاقة(Supercomputers) هذه الأيام تعتمد بشكل كـبير علـى التوازي . وهى تُستخدم في مستوى البرمجيـات وكـذلك التـصميم الهندسـي للعتـاد (Hardware). واشتد السباق بين دول العالم ولكي تتنافس في مجـال الاقتـصاد فالبلـدان تتطلب اكتشافات علمية ومهندسين وعلماء حاسب ليقوموا بتوظيف الحاسبات العملاقـة بشكل سليم.

لقد استخدمنا عبارة "المعالجة المتوازية" مرات عديدة في هذا السياق وحان الوقت للتمييز بين هذه العبارة وغيرها وخاصة " البرمجة المتوازية"(انظر الفصل الرابع)، و لأن مجال علـوم الحاسب تعتريه مشكلة المصطلحات. ولا يوجد هناك تعريف موحد للمعالجة المتوازية ولذلك سوف نوضح كيف سنستخدم هذا اللفظ.

المعالجة المتوازية(Parallel Processing): **معالجة الحاسب الآلي لعدة برامج في آن واحد(سويّا) وباستعمال عدة وحدات معالجة حسابية ومنطقية.**

وتعتبر المعالجة المتوازية حقلاً جزئياً من علم الحاسب والتي تتضمن مفاهيمَ و أفكاراً من علوم الحاسب النظرية و هندسة الحاسب و لغات البرمجة و الخوارزميات ومجالات التطبيـق مثل الذكاء الاصطناعي والرسوم.

1.5 تطبيقات المعالجة المتوازية

تحتاج التطبيقات في العديد من المالات إلى قدرة حسابية عالية و تتزايد هذه الحاجة مع الاعتماد بشكل أكبر على الحاسبات في مجالات الصناعة و الزراعة والتحكم والدراسـات المختلفة . ونلمس أهمية الحساب السريع و بالتالي المعالجة المتوازية في مجالات متعددة نـذكر منها :

النمذجة والمحاكاة : تنبؤات الأرصاد الجوية ، علم المحيطات ، الفيزياء الفلكية .
الهندسة : ميكانيك الموائع ، الهندسة النووية ، الهندسة الكيميائيـة الربوتيـة ، الـذكاء الاصطناعي ، معالجة الصور ، وغير ذلك.
البحث عن مصادر الطاقة : الكشف عن البترول و المعـادن ، الكـشف الجيولـوجي ، البحث الطـبي و العسكري .

تقاس القدرة الحسابية للحاسوب بعدد العمليات التي يمكن أن ينفذها بالثانية على الأعداد الحقيقيـة((Mega FLOPS (Floating Point Operation Per Second). و تتطلـب التطبيقات السابقة قدرة حسابية من درجة ألف مليون عملية في الثانية.لقد وصـلت قـدرة الحاسبات المتفوقة في عام 2003 إلى 35 TFLOPS وهو ما يقـارب 36.5 تريليـون عمليـة حسابية كل ثانية. ومن أجل تحسين الأداء يجب تقصير زمن دورة المعالج للعمليـة الواحـدة

وزيادة عدد العمليات التي تنفذ على التوازي. ولكن تـسريع المعالجـات لا يكفـي إذا لم يتصاحب بالقدرة على نقل المعلومات من الذاكرة إلى وحدة المعالجة بالسرعة الكافية و تقاس هذه القدرة بعدد البايتات Bytes التي يمكن نقلها من الذاكرة إلى وحدة المعالجة في الثانية .

1.6 تعريف الحاسب المتوازي

إذا قلنا بأن الحاسب المتوازي هو الذي يستخدم التوازي فإننا نشمل جميع أجهزة الحاسب الموجودة! وجميـع الحاسـبات تـستخدم التـوازي علـى مـستوى بـسيط في طـرق البيانات(datapaths). ولكننا نقول بأن بعض الآلات متوازية والبعض الآخر غير متوازيـة. ونحتاج إلى اختبار لكي نفرق بينها، وسنضع اختباراً ليس دقيقاً ولكنة ذو علاقة بالاستخدام العام. فإذا استطاع المستخدم أن يكتب برنامجا يمكنه أن يقرر فيما إذا كانت الصفة الهندسية المتوازية موجودة و القول بأن التوازي مرئي.

الحاسب المتوازي (Parallel computer)**:**

هو حاسب يستخدم عدة معالجات تعمل بشكل متزامن (أي تعمل في نفس الوقت) لحل مسألة أو لأداء وظيفة معينة.

ويمكن للبرمجيات التي تُكتب للحاسب المتوازي أن تزيد كمية العمل المنفذ في مدة محددة من الوقت وذلك بتقسيم المهام الحسابية بين عدة معالجات تعمل بشكل آني.

1.7 التسريع (Speedup)

إن التكنولوجيا تُحسن باستمرار أداء المعالجات المفردة. والبشر بارعون في إيجاد المسائل التي يبدو من أجلها المعالج المفرد بطيئاً. وباستخدام عدة معالجات على التوازي هنالك أمل في حلها بشكل سريع، وبشكل مثالي فإنه يمكن الحديث عن التسريع وفقاً للتعريف الآتي:

التسريع (S_p): **هو حاصل قسمة الزمن التسلسلي (t_s) على الزمن المتوازي (t_p).**

ومن ذلك نكتب : $S_p = t_s / t_p$

حيث أن : S_p : التسريع.

t_s: الزمن التسلسلي .

t_p: الزمن المتوازي.

يمكن توضيح مفهوم التسريع من خلال المثال التالي المبسط :

nفلو أن عامل البناء يحتاج إلى وحدة زمن واحدة لبناء جدار، فما هو الزمن اللازم لـ عامل في هذه المهمة . لنفرض أن الحالة مثالية وهي أن عمال البناء لا يعيق بعضهم البعض ، وحدة زمن . ولقد لاحظنا 1/nففي هذه الحالة يجب على البنائين أن ينهوا بناء الجدار خلال بأن:

$$\text{التسريع} = \frac{\text{الزمن التسلسلي}}{\text{الزمن المتوازي}}$$

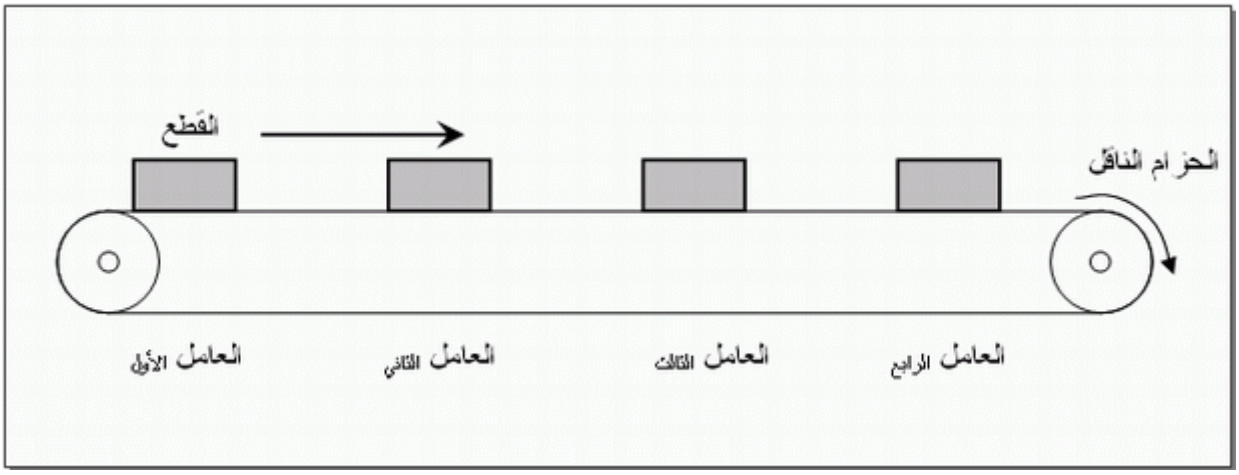
والزمن التسلسلي هنا لعامل البناء الواحد.

والزمن المتوازي هو الزمن لـ n عامل بناء.

وبالتالي فإن نسبة التسريع في هذه الحالة: $S_p = 1/(1/n) = n$

وبناءً على افتراضنا السابق، فإن n من عمال البناء أسرع n مرة من العامل الواحد.
وفي الحالة العامة فإنه مع n ضعف تكون إمكانية التسريع إلى n مره.

لنفترض وجود خط تجميع (تجميع أجزاء منتج مثلاً) بحيث أن إنتـاج قطعـة واحـدة يستلزم أداء أربعة مهام، كل مهمة يؤديها عامل واحد .



الشكل (1-1) :خط تجميع مكون من أربع محطات عمل

و لنفترض أيضاً أن المهام الأربع تحتاج إلى T وحدة زمن، ففي وحدة الزمن الأولى سينفذ العامل الأول المهمة الأولى من القطعة الأولى، وفي وحدة الزمن الثانية سينفذ العامـل الأول المهمة الأولى من القطعة الثانية وسينفذ العامل الثاني المهمة الثانية من القطعة الأولى وهكذا...

بعد أربع وحدات زمن سيكتمل إنتاج القطعة الأولى وستخرج من خط الإنتاج، وبعد ذلك سيكتمل إنتاج قطعة واحدة في كل وحدة زمن. دعنا نفترض أننا نريد إنتاج عشر قطع، فما هو التسريع الحاصل في خط الإنتاج بالنسبة لعامل واحد يؤدي كل المهام الأربع؟

إن الزمن لعامل واحد (T_{W1}) هو :

$$T_{W1}=10*4*T$$
$$=40T$$

حيث : T : وحدة الزمن .
سيحتاج العامل لـ 4T لبناء كل قطعة وبالتالي سيحتاج إلى 40T لبناء العشر قطع.
أما في نظام التجميع، فنحتاج لإكمال إنتاج القطعة الأولى إلى 4T ... وباقي القطع ستحتاج إلى 1T.

إذن إن زمن خط التجميع (T_L) يحسب بالعلاقة :

$$T_L = 4*T+(10-1)*T$$

وبما أن :

$$\textbf{التسريع لخط التجميع} = \frac{\text{الزمن لعامل واحد}}{\text{الزمن لخط التجميع}}$$

فإن التسريع لخط التجميع هو :

$$\frac{T_{W1}}{T_L} = \frac{40\ T}{13\ T} = 3.07$$

ولإنتاج عشر قطع فالعمال الأربعة في خط التجميع تقريباً أسرع بثلاث مرات من العامـل الواحد .

لنفترض أيضاً أننا نريد إنتاج k قطعة : إذن :

$$T_{W1} = T * 4 * k$$

وبالتالي فإن : $T_L = 4 * T + (k-1) * T$

إن السلوك أو السير التقريبي للتسريع عندما ننتج العديد من القطع أو عندما تقترب k من ∞ يمكن متابعته من خلال المناقشة الرياضية الآتية فإن التسريع لخط التجميع (S_p) كالآتي :

$$S_p = \frac{k * 4 * t}{4 * t + (k-1) * t} = \frac{4 * k}{3 + k}$$

وبقسمة البسط والمقام على k نستنتج أن :

$$S_p = \frac{4}{\frac{3}{k} + 1}$$

وعندما تسعى K إلى اللانهاية فإن التسريع لخط التجميع يساوي إلى 4 ، لذا فإنه في خط تجميع ذو أربعة محطات، فإن التسريع التقريبي هو أربعة.

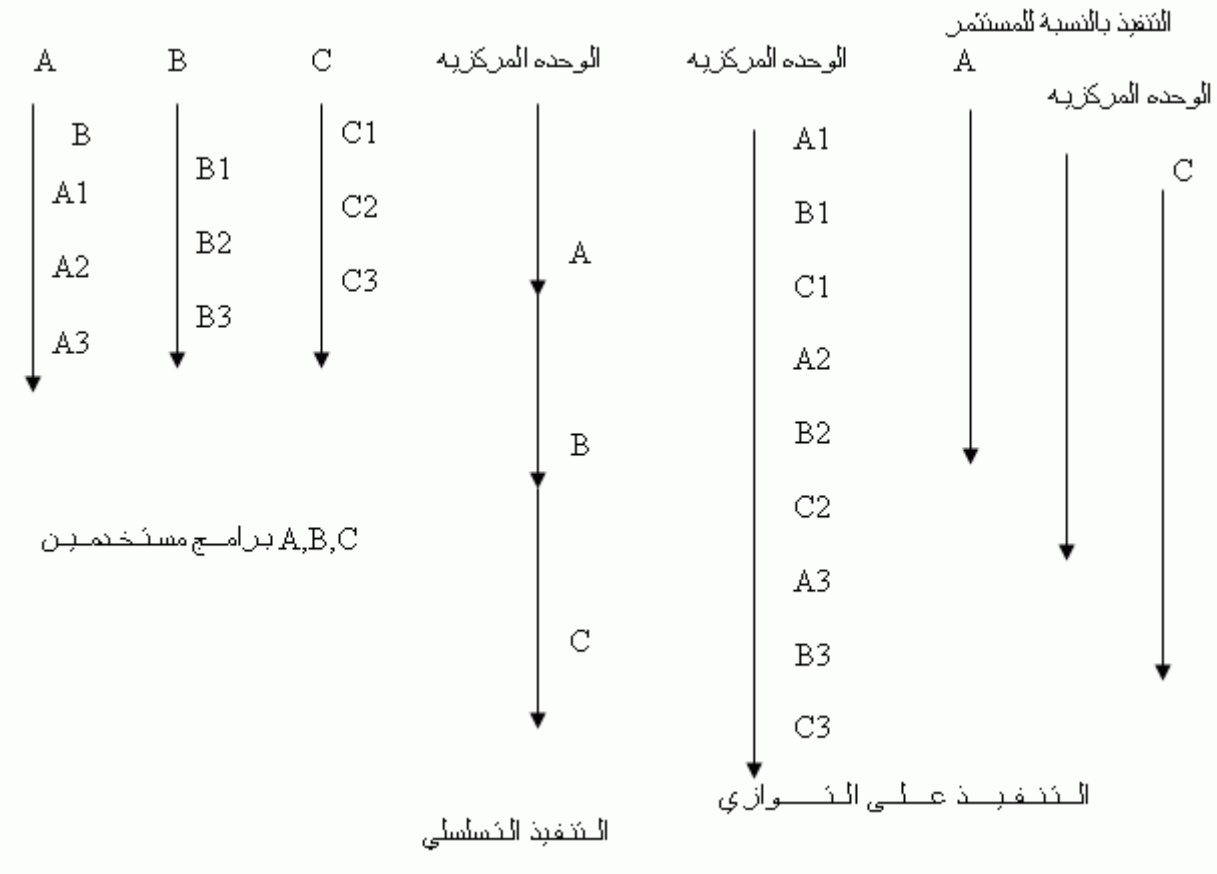
وبشكل عام: بافتراض تساوي الزمن في كل محطة فإن n محطة في خط التجميع تقريباً سيكون لها تسريع يساوي (n).

1.8 أشكال معالجة المعطيات على التوازي

المعالجة المتوازية هي شكل من أشكال معالجة المعطيات تسمح بتنفيذ عدد من الأحـداث المتزامنة بنفس الوقت . هذه الأحداث المتوازية يمكـن أن تكـون على مستويات مختلفة :

1.8.1 مستوى البرامج (Programs)

يتم تنفيذ عدد من البرامج المستقلة عن بعضها بنفس الوقت و تستخدم مبادئ تعددية البرمجيات (Multiprograms) و المشاركة الزمنية (Temporal Partcipation) وتعددية المعالجة (Multiteatment) من أجل تحقيق ذلك .



الشكل (2-1): تعددية البرمجيات

يستثمر هذا المستوى على الحاسبات الكبيرة و تكون المعالجة على التوازي شـفافة بالنـسبة للمستخدم حيث يتولى نظام التشغيل مهمة إدارتا و تطويرها . يمكن بالاعتماد على مبـدأ تعددية البرمجيات ، تنفيذ عدة برامج لمستخدم واحد أو عدة برامج لمـستخدمين مخـتلفين . وتعتمد إحدى طرق تعددية البرمجيات على تقسيم زمن الوحدة المركزية إلى مجالات زمنيـة متساوية بحيث تشغل الوحدة المركزية برامج مختلفة بشكل دوري خلال الـمجالات الزمنيـة المختلفة، كما هو موضح في الشكل (2-1) ، ففي التنفيذ التسلسلي تنفيذ العملية الأولى (A) ومن ثم العملية الثانية (B) وأخيراً تنفيذ العملية الثالثة (C) . أما التنفيذ على التوازي فُتقـسم العمليات الثلاث إلى أجزاء زمنية متساوية و تنفذ على التوازي بحيث تنفذ العمليات علـى الترتيب التالي :A1,B1,C1,A2,B2,C2,A3,B3,C3 .

1.8.2 مستوى الإجرائية (Procedure)

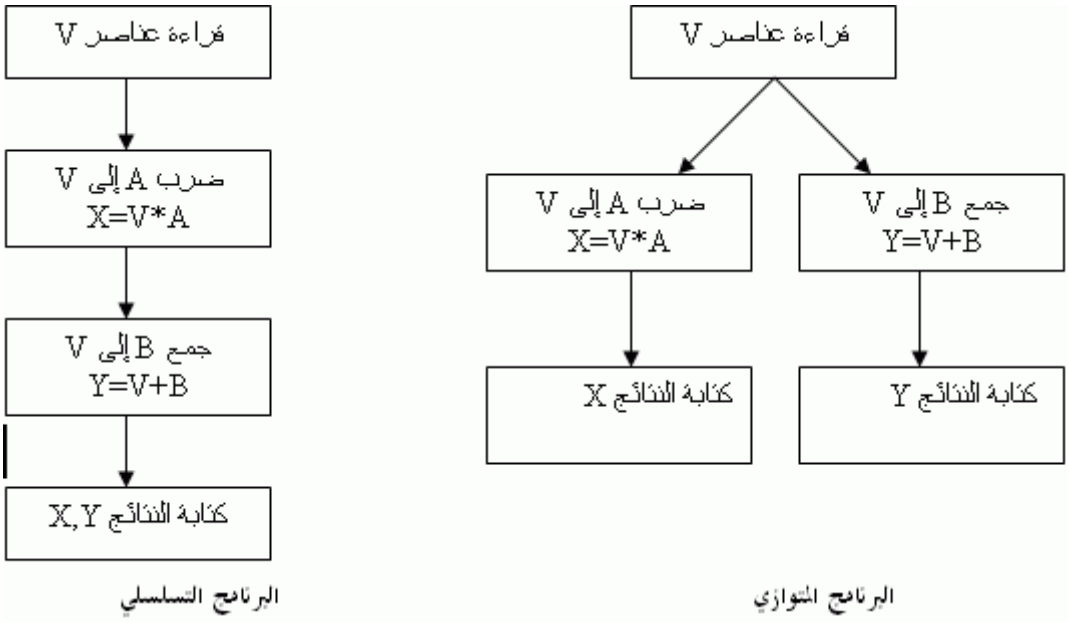
يتطلب هذا المستوى تقسيم البرنامج الواحد إلى عدة مهمات على التوازي. إن تقـسيم البرامج على هذا الشكل ليس بالعمل السهل فمهمات البرنامج الواحد مرتبطة فيما بينـها و غالباً ما يعتمد تنفيذ مهمة ما على نتائج المهمات الأخرى لذا لابد من البحث عن علاقـات التبعية ومن ثم برمجة المهمات غير المرتبطة أو المرتبطة جزئياً على التوازي.

يقوم نظام التشغيل بإدارة ومعالجة هذا المستوى من التوازي إذا تم اسـتخدام أدوات آليـة للتوازي أو مترجمات ذكية ولكن هذه الأدوات ما زالت في مجال البحث ولم تثبت فعاليتـها بعد، وبالتالي فغالباً ما يقوم المستخدم بتحليل البرامج على مستوى الخوارزميات و تحويلها إلى برامج متوازية.

يبين المثال التالي هذا المستوى من التوازي :
يقوم أحد البرامج بتنفيذ عمليتين على الأشعة ، الأولى : ضرب عناصر الشعاع بثابت (a) و

الثانية جمع ثابت (b) إلى عناصر الشعاع. و بالتالي يقوم بالإجرائيات التالية كما في الـشكل (1-3):

- قراءة عناصر الشعاع V.
- ضرب عناصر الشعاع V بالعدد a و تخزين النتائج في الشعاع X .
- جمع b إلى عناصر الشعاع V و تخزين النتائج في الشعاع Y.
- طباعة النتائج : الشعاع X والشعاع Y.



الشكل (3-1) : تحويل برنامج تسلسلي إلى متوازي (مستوى الإجرائيات)

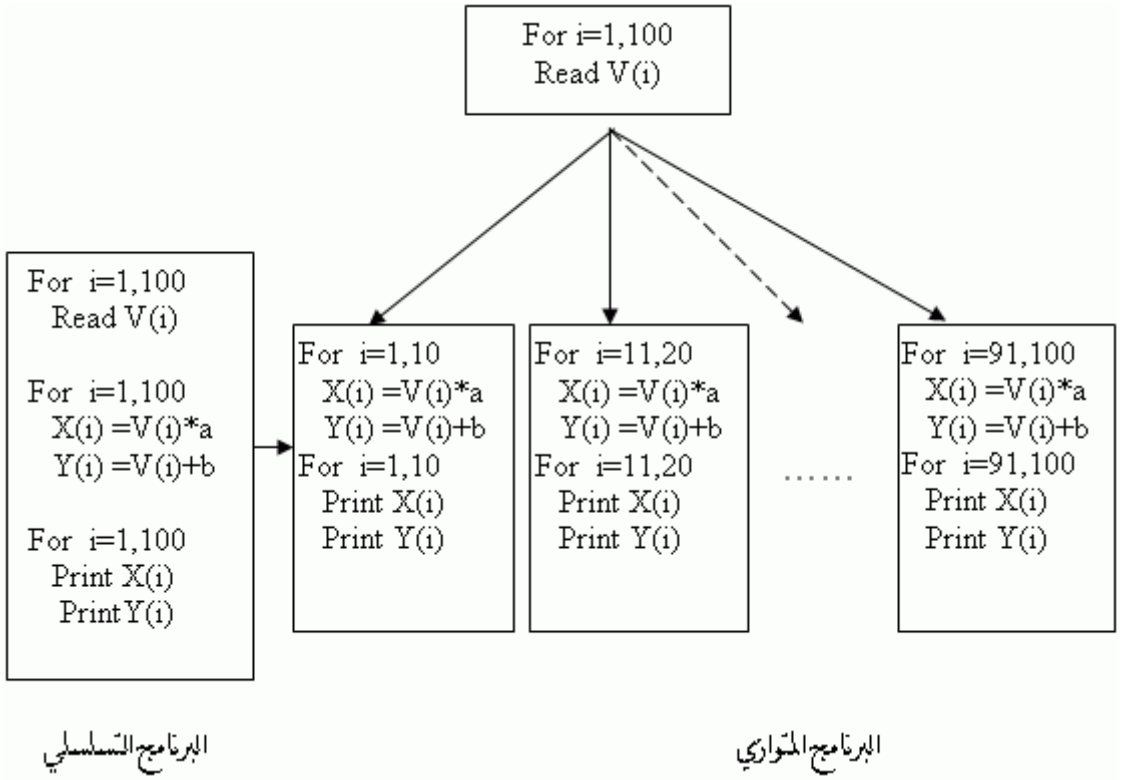
1.8.3 مستوى التعليمات (Instructions)

يوجد العديد من التقنيات التي تعتمد مبدأ تنفيذ عدة تعليمات مستقلة فيما بينها علـى التوازي و أشهرها تقنية معالجة الجداول التي تقوم بتنفيذ تعليمة وحيدة تعالج معطيات مختلفة على عدة معالجات بنفس الوقت .

يعالج هذا النوع من التوازي على مستوى :

- نظام التشغيل نظراً لتوفر الأدوات الآلية المساعدة على تحويل البرامج .
- اللغات البرمجية حيث تتوفر على الحاسبات الشعاعية لغات برمجية خاصة للبرمجة الـ شعاعية مثل : FORTRAN Victories .

يستخدم هذا النوع من التوازي في برامج الحساب العملي بشكل خـ ـاص حيث تعالج الأشعة و المصفوفات . و يبين الشكل (4-1) تحويل برنامج تسلسلي إلى متوازي على مستوى التعليمات .



الشكل (4-1) : تحويل برنامج تسلسلي إلى متوازي (مستوى التعليمات)

1.8.4 مستوى التعليمة (Instruction)

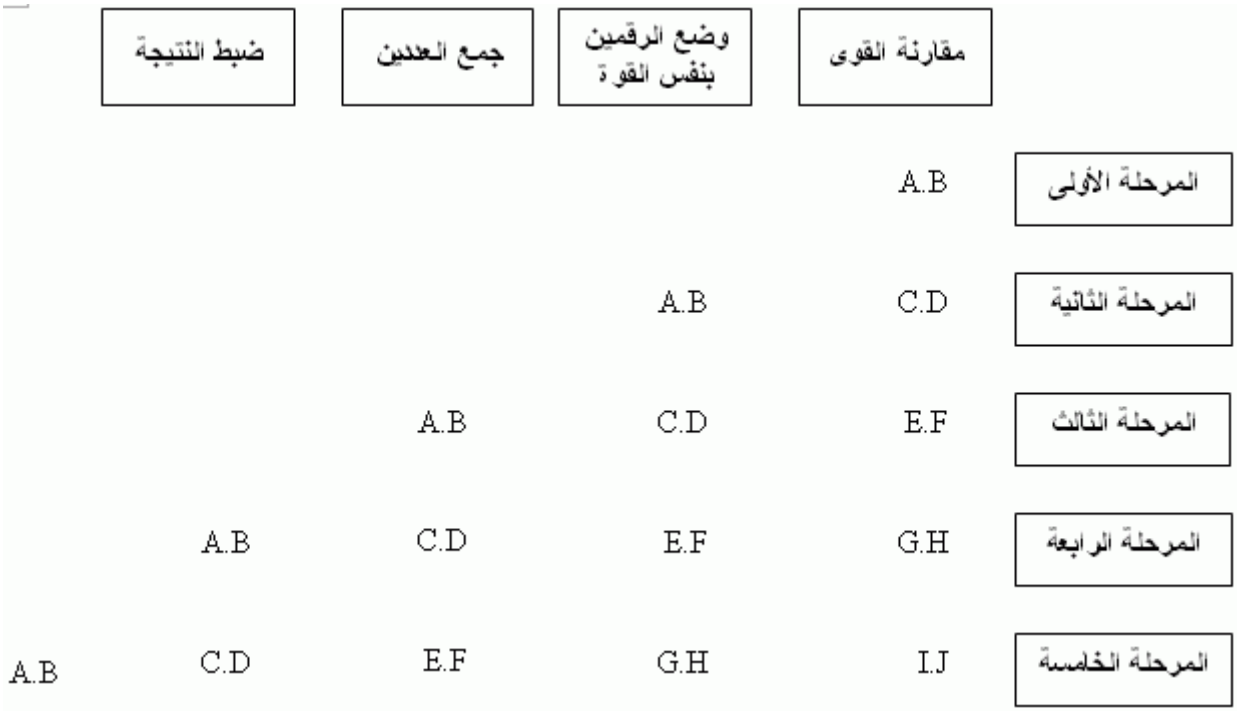
تعتمد تقنيات العمل الضخي (Pipeline) مبدأ تقسيم التعليمة الواحدة إلى تعليمات جزئية متتالية بحيث يمكن تنفيذ هذه التعليمات الجزئية بنفس الوقت على معطيات مختلفة.

يتم الاستفادة من هذا النوع من التوازي على مستوى العتاد، وهـو شـفاف بالنـسبة للمستخدم في الحاسبات الحالية. و تستخدم أغلـ ـب المعالجات والبطاقات المتخصصة ذات الأداء العالي في يومنا هذا مبدأ العمل الضخي في تصميمها. يبين المثال التالي هذا المستوى من التوازي:

يمكن تقسيم عملية ضرب عددين مثلين بالفاصلة العائمة إلى العمليات الجزئية التالية :

١- مقارنة القوى.
٢- وضع الرقمين بنفس القوة.
٣- القيام بعملية جمع العددين.
٤- كتابة النتيجة بشكل مضبوط.

وباعتبار هذا التقسيم يمكن إجراء عدة عمليات ضرب على سلسلة من الأعـداد بـنفس الوقت، وذلك بإجراء المراحل المختلفة على ثنائيات أعداد مختلفة. ويوضح الشكل(5-1) طريقة إجراء العملية على التوازي.



الشكل(5-1): العمل الضخي (Pipeline)

وأخيراً تجدر الإشارة إلى أن الحاسبات المتوازية المتقدمة تستثمر المستويات الأربعة من العمل على التوازي.

1.9 موجز لتاريخ الحاسبات

إنّ التاريخ الكامل للحاسبات والحوسبة يجب أن يتطرق لعدد كبير من الآلات المتنوعة مثل المعداد (Anacuy) الذي كان يستخدمه الصينيون القدماء، كذلك الآلة التحليلية التي اخترعها "باباج" و "لووم"، أيضاً يجب أن يتطرق للعمارة الرقمية والتماثلية(analog and digital) للحاسبات.

إن تطور الحاسبات الرقمية غالباً يُقسم إلى أجيال. كل جيل يتميز بتطور ملحوظ عن سابقه في التقنية المستخدمة لبناء الحاسبات، كذلك التنظيم الداخلي لنظم الحاسب، ولغات البرمجة. وبالرغم من أن الخوارزميات عادة لا تؤخذ بعين الاعتبار في أجيال الحاسبات، إلا أنا

أيضاً تتطور وبثبات، ويشمل ذلك الخوارزميات المستخدمة في العلوم الحاسوبية. التاريخ التالي سيعتمد على هذه الأشياء كمعالم لتقسيم الأجيال.

العصر الميكانيكي (1623-1945)

إنّ فكرة استخدام الآلات لحل المسائل الرياضية يمكن إرجاعها إلى أوائل القرن الـسابع عشر الميلادي. الذين صمموا ونفذوا الآلات الحاسبة التي يمكن لها أن تؤدي عمليات الجمع والطرح والضرب والقسمة هؤلاء العلماء هم "Schickhard" و "Pascal" و "Leibnitz".

إن أول آلة حسابية متعددة الأغراض هي آلة "بّاباج" المـسماة بـ ـ "آلـة الفـرق" أو "Difference Engine"، والتي بدأ بها في عام 1823 ولكنها لم تكتمل. آلة أخـرى طموحـة وهي الآلة التحليلية والتي بدأ باباج بتصميمها في العام 1842 ولكنها لم تكتمل أيضاً.

الجيل الأول للحاسبات الآلية

فترة الجيل الأول من أول ظهور للحاسب في منتصف الأربعينات إلى أن أنتج الحاسـب IBM 650 (1956).

- كانت الحاسبات في الجيل الأول تستخدم الصمامات المفرغة كعناصر تحويل أساسـية، وتستخدمها أيضاً للذاكرة.
- قدرة المعالجة لتلك الحاسبات المبكرة تقدر بـعشرة آلاف تعليمة كل ثانية.
- أما عن التخزين فكان بإمكان تلك الحاسبات تخزين ٢٠٠٠ حرف أبجدي أو رقمي.
- **ظهر** في **هذا الجيل آلة "فون نيومان" IAS كانت الأولى** في **توظيف علـم الحـساب المتوازي، والتي بدأ "فون نيومان" بتصميمها** في **عام 1946، ولم تكتمل بجميع وظائفها إلا** في **عام 1952.**
- **ظهر** في **هذا الجيل أول آلة** تجارية **تستخدم الحساب المتوازي وهي الآلة** IBM 701.

• كان الحاسب الآلي في هذا الجيل تسلسلي العمارة، وفيه طرحت اقتراحات لآلات متوازية وبدرجات متفاوتة، ولكنها لم تكن تتجاوز مرحلة التنميط (النمذجة).

• كانت البرامج في أول الجيل تكتب بلغة الآلة، ومع بدء الخمسينات بدأ استخدام لغة التجميع وكانت الترجمة من لغة التجميع إلى لغة الآلة تتم يدوياً، وبعد ذلك تم عمل المجمع Assembler والذي يقوم بعملية تحويل البرامج من لغة التجميع إلى لغة الآلة.

• من الحاسبات المشهورة في الجيل الأول :

- ENIAC واستُخدم حينها للحسابات الخاصة بالقنبلة الهيدروجينية التي ألقيت على هيروشيما وناجازاكي.

- EDVAC

- UNIVAC والذي استخدم للتنبؤ بنتيجة الانتخابات الرئاسية للولايات المتحدة الأمريكية لعام 1952.

الجيل الثاني

تمتد فترة هذا الجيل من عام 1957 وحتى عام 1963، في هذا الجيل حدثت تطورات هامة جداً على كل المستويات من بناء الدارات الأساسية إلى لغات البرمجة، ومن السمات المميزة له:

• استُخدِم في هذا الجيل الترانزستورات بدلاً من الصمامات المفرغة.

• قدرة المعالجة لحاسبات الجيل الثاني تقدر بـمائتي ألف تعليمة كل ثانية.

• يمكن لحاسبات هذا الجيل تخزين 32000 حرف أبجدي أو رقمي.

• تم في هذا الجيل إنتاج العديد من لغات البرمجة العالية المستوى، مثل FORTRAN (1956)، و ALGOL(1958)، و COBOL(1958).

• من الآلات التجارية الهامة في الجيل الثاني هي الآلة IBM 704 وكان فيها وحدة للنقطة العائمة الحسابية. جاعلاً منها وبنجاح أسرع من الآلات السابقة. **وهي الآلة الأكثر نجاحاً** في **استخدام الحساب المتوازي** في **وقتها.**

• ومن الآلات الهامة في هذا الجيل الآلة IBM 7094، وفيها استخدمت معالجات دخل/خرج (I/O) ليتم التواصل بين الذاكرة الرئيسية وأدوات الإدخال والإخراج بشكل أفضل.

• شهد هذا الجيل ظهور أول حاسبين عملاقين صمما خصيصاً لمعالجة الأعداد في التطبيقات العلمية. الحاسبان هما LARC و IBM 7030 (ويطلق على الأخير أيضاً Stretch). **و كان الحاسب IBM 7030 أول حاسب يستخدم التوازي** في **الذاكرة،** وذلك لتمكين الذاكرة المغناطيسية البطيئة للتماشي/التوافق مع المعالجات الأسرع.

الجيل الثالث

تمتد فترة هذا الجيل من عام 1964 وحتى منتصف السبعينات، ويمكن تلخيص و تحديد أهم المعالم لهذا الجيل في الآتي :

• استُخدم في هذا الجيل الدوائر المتكاملة Ics (الدوائر المتكاملة هي أشباه موصلات وعدد من الترانزستورات جمعت في قطعة واحدة)

• **ظهر** في **هذا الجيل أول حاسب متوازي، وهو الحاسب Illiac IV عام (1972).**

• قدرة المعالجة لحاسبات الجيل الثالث تقدّر بخمس ملايين تعليمة كل ثانية.

• يمكن لحاسبات هذا الجيل تخزين مليوني حرف أبجدي أو رقمي.

• استُخدم للذاكرة أشباه الموصلات بدلاً من المغناطيس.

• ظهرت في هذا الجيل تقنية فعالة لتصميم المعالجات المعقدة وهي تقنية (microprogramming)

- ظهرت نظم التشغيل.
- **بدأ مصممو الحاسبات بالاستفادة من التوازي وذلك باستخدامهم لوحدات وظائفية متعددة.**
- **كان الحاسب CDC 6600 من الحاسبات الهامة** في **أوائل هذا الجيل (1964) وكـان أول حاسب يستخدم التوازي الوظيفي**، فكان له عشر وحدات وظيفية يمكنها العمـل في نفس الوقت.
- وفي هذا الجيل ظهر أول حاسب يستخدم معالجات موجهة (Vector processor) وهو الحاسب CDC 7600 (أنتج في 1969).
- من الحاسبات الهامة في هذا الجيل CRAY 1, IBM 360 and 370 series ,CYBER 205
- في عام 1972 أنتجت لغة البرمجة C، أيضاً نظام التشغيل "يونيكس".

الجيل الرابع

تمتد فترة هذا الجيل من منتصف السبعينات وحتى أواخر الثمانينات، نلخص أهم معـالم هذا الجيل في الآتي:

- استُخدم في هذا الجيل الدوائر المتكاملة الواسـعة (Large scale integration) كمـا استخدم في هذا الجيل المعالج الدقيق (Micro-processor)
- من الحاسبات الهامة في هذا الجيل:
  - **CRAY X-MP نظام بأربعة معالجات.**
  - CRAY 2 أول حاسب يتمكن من تنفيذ مليار عملية حسابية في الثانية.
  - **CYBERplus و استخدم فيه نظام عدة معالجات متوازية** (٦٤ معالج عالي الأداء)

أجيال المستقبل

يبدأ هذا الجيل من أوائل التسعينات ،وبدأت الأبحاث تتجه في اتجاهين في محاولة لمحاكاة العقل البشري. الاتجاه الأول يحاول تمثيل الحاسوب كشبكات عصبية وهـو مـا يعـرف بالشبكات العصبونية الاصطناعية(Artifical Neural Network)، والاتجاه الآخر يحـاول بالتعاون مع علماء الهندسة الوراثية، في إنتاج رقاقة حيوية وذلك بتكييف البر وتينات لتحـل محل السيليكون في الدوائر الإلكترونية. وتخدم أبحاث هذين الاتجاهين أساس الجيـل القـادم للحواسيب.

ويمكن إيجاز الأبحاث التي تجري من أجل حواسيب المستقبل في مجموعة من الاتجاهات أو المحاور. أول هذه الاتجاهات يختص بالمكونات المادية(Hardware)، وثاني هذه الاتجاهات هو أساليب **العمل على التوازي** والاتصالات، وثالث هذه الاتجاهات يخـتص بالبرمجيـات (Software).

ففي مجال المكونات المادية تتزايد إمكانيات وسرعات المعالجـات المنتجـة. وفي مجـال المكونات المادية غير الفعالة مثل الذاكرة تتزايد أحجام الذاكرة في الرقاقة الواحدة.

وفي مجال أساليب **العمل على التوازي** والاتصالات فقد تبين إمكـان تنفيـذ ملايـين التعليمات في الثانية الواحدة وذلك عن طريق استخدام أكثر من معالج. وتـبين أن تعـاون المعالجات في تنفيذ التعليمات يكون أيسر إذا كانت تعليمات هذه المعالجات بسيطة.

وفي مجال البرمجيات وصلت بعض البرامج الخاصة بالجيـل الحـالي مثـل بـرامج الـنظم الخبيرة(Expert System) إلى مرحلة النضوج. ومازالت هناك حاجة إلى المزيد من العمل في مجالات الذكاء الاصطناعي لتطوير وسائل إدخال البيانات والتـساؤلات إلى الحاسـوب بطريقة طبيعية أكثر مثل التخاطب مع الحاسوب بلغة طبيعية والكتابة بأساليب بسيطة.

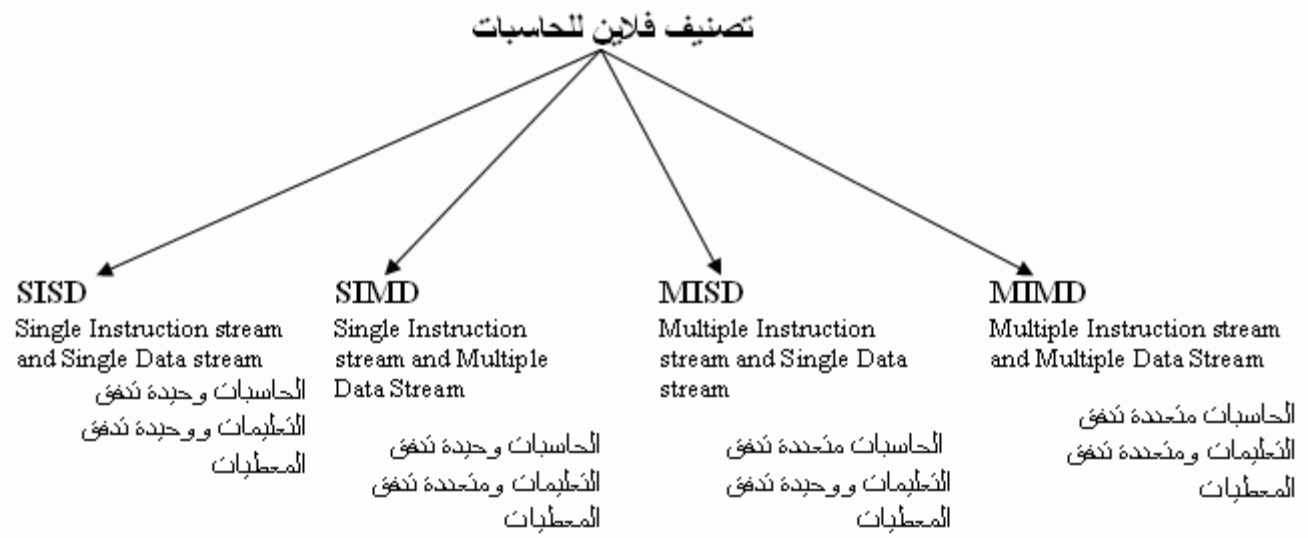
الفصل الثاني: تصنيف الحاسبات المتوازية

تم في العقدين الأخيرين تصميم وإنشاء الكثير من الحاسبات المتوازية. ويمكن أن نصنفها في مجموعات اعتماداً على السمات المشتركة. وتمكننا خطة التصنيف هذه من دراسة آلـة أو أكثر كنموذج لكل مجموعة مما يساعدنا على فهم أفضل للمجموعة. ولسوء الحظ، لم يجـد الباحثون مخطط تصنيف مقنع يمكن أن يغطي كل أنواع الآلات المتوازية. وعلى مر الـسنين، كان هناك العديد من المحاولات لإيجاد طريقة فعالة ومريحة لتصنيف الحاسبات مـن حيـث العمارة. وعلى الرغم من أنه ليس هناك تصنيف كامل، إلا أن التصنيف الأكثر انتشاراً هذه الأيام هو ذلك التصنيف الذي اقترحه "ميشيل فلاين" [Flynn] في عام 1966. ويأخذ تصنيف فلاين بعين الاعتبار عاملين اثنين وهما: كمية السريان (أو التدفق) للتعليمات و كمية السريان للبيانات التي تتدفق للمعالج. ونورد هذا التصنيف في الفقرة الآتية:

2.1 تصنيف فلاين [Flynn's Classification Scheme]

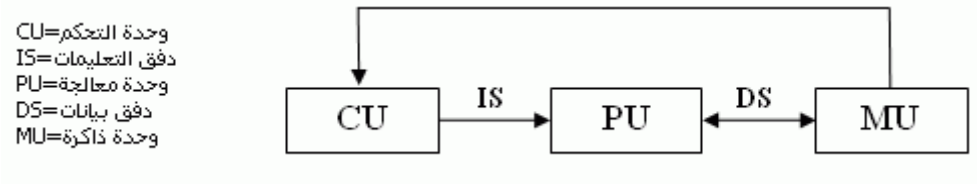
إن تصنيف فلاين مبني أساساً على كمية تدفق البيانات والتعليمات الموجودة في الآلـة. ويقصد بالتدفق هنا على أنه تتابع أو تسلسل لعناصر (التعليمات أو المعطيات) كما نفذت أو شغلت بواسطة المعالج. فعلى سبيل المثال، تقوم بعض الآلات بتنفيـذ تـدفق واحـد مـن التعليمات، بينما يتم تنفيذ عدة تدفقات في آلات أخرى. وبنفس الطريقـة فـبعض الآلات ترجع تدفقاً واحداً من المعطيات، وآلات أخرى ترجع تدفقات متعددة. وعليه فإن فلايـن

يضع الآلة في تصنيف واحد من بين أربعة وذلك اعتماداً على وجود تدفق واحد أو تدفقات متعددة.



2.1.1 الحاسبات وحيدة تدفق التعليمات ووحيدة تدفق المعطيات SISD

ويندرج تحت هذا الصنف جميع الحاسبات التسلسلية المعتادة، مثل "Apple Macintosh" و "DEC VAX". والحاسبات من هذا الصنف تُحضر التعليمات من الذاكرة ثم تقوم بتنفيـذها عادة باستخدام قيم المعطيات المشار إليها من الذاكرة. ومن ثم تقوم بإحضار تعليمات أخرى من الذاكرة، وهكذا. و أيضاً يعرف هذا التصميم (SISD) بتصميم "فـون نيومـان Von Neumann" والذي قام به العالم "جون فون نيومان" في أواخر الأربعينات وأوائل الخمسينات من القرن الماضي. لقد مرت صناعة الحاسبات وعلى مدى خمسين عاماً بالكثير من الخبرة في هذا التصنيف فالكثير من لغات البرمجة (مثل Fortran و C)، والمترجمات، ونظم التـشغيل، وطرق البرمجة، كلها تعتمد أساساً على هذا التصنيف.



الشكل (2-2): SISD

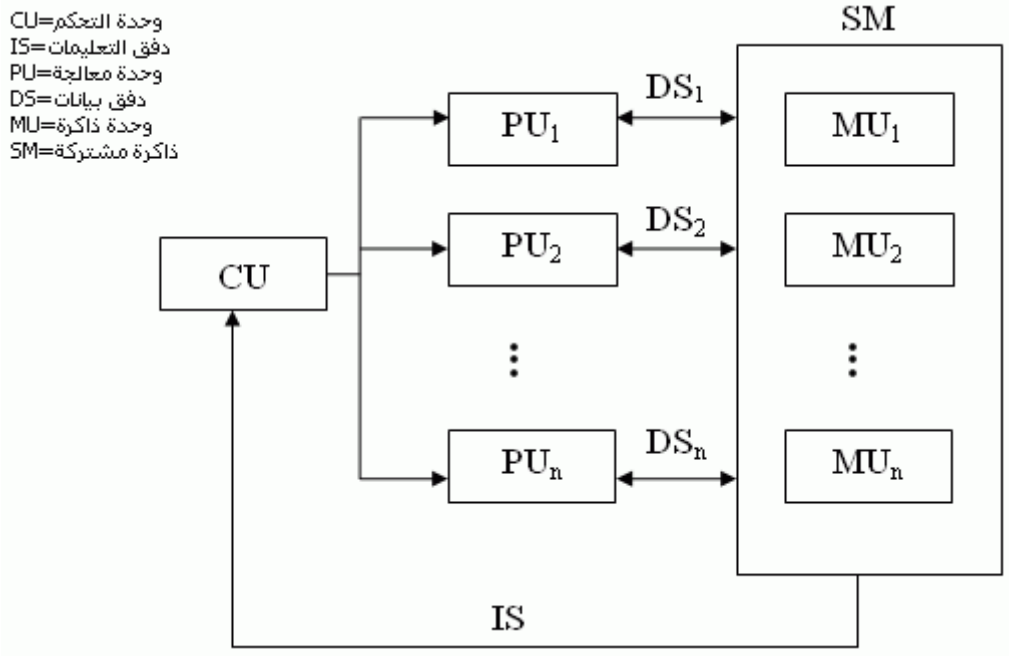
إن كل أنواع الحاسبات التسلسلية موجودة في تصنيف SISD، بل وأكثر من ذلك؛ فالباحثون وضعوا بعض أنواع الحاسبات المتوازية ضمن هذا التصنيف، فعلى سبيل المثال، المعالجات الشعاعية (Vector) مثل Cray-1 تنتمي إلى هذا التصنيف بالرغم من أن تعليمات التشغيل تعمل على قيم بيانات شعاعية إلا أن لديها دَفقٌ واحد من التعليمات.

2.1.2 الحاسبات وحيدة تدفق التعليمات ومتعددة تدفق المعطيات SIMD

ويتضمن هذا التصنيف الحاسبات التي تحتوي على وحدة تعليمات واحدة تصدر أوامر إلى عدة عناصر معالجة (PEs). ولأن كل عنصر معالجة (PE) يُشغّل بياناته المحلية الخاصة فهناك تدفقات متعددة للبيانات. وعادة فإن وحدة التعليمات تصدر نفس الأمر إلى جميع عناصر المعالجة . على سبيل المثال جميع عناصر المعالجة تنفذ تعليمة الجمع ADD، وبعد ذلك تنفذ تعليمة التخزين STORE، وهكذا.

يتميز هذا النوع من الحاسبات بوجود وحدة تحكم مركزية . وتشرف على عناصر المعالجة المختلفة تعليمة واحدة من وحدة التحكم وتقوم بتنفيذ هذه التعليمة بشكل متزامن على معاملات مختلفة. تكون هذه الحاسبات متزامنة، وغالباً ما تملك ذاكرة مشتركة بين الوحدات. ولتسهيل عملية الولوج المتوازي إلى الذاكرة يُلجأ إلى تقسيمها إلى بنوك مما يسمح باستخلاص عدة معاملات في نفس الوقت، تتبادل وحدات معالجة المعطيات عن طريق الذاكرة المشتركة ويتم الاتصال بين وحدات المعالجة المختلفة وبنوك الذاكرة عبر شبكة الربط. ونظراً لتنفيذ نفس العملية على الوحدات المختلفة فمن الممكن اعتبار الحاسبات

SIMD كحاسوب وحيد المعالج الذي يقوم بتنفيذ التعليمات على أجزاء مختلفة من المعطيات. ويلائم هذا النوع من الحاسبات العمليات على الأشعة وعلى المصفوفات وغالباً ما يستخدم من أجل عمليات الحساب العلمي. ويوضح الشكل(2-3) هذا الصنف حيث تؤخذ المعطيات من الذاكرة وينفذ عليها أمر واحد، ومثال على ذلك عملية ضرب العدد B بالشعاع A(I) حيث I=0,…,N، والناتج يوضع في C(I)، فتحصل عملية ضرب B بجميع عناصر الشعاع A(I) بعملية واحدة على كل المعاملات.

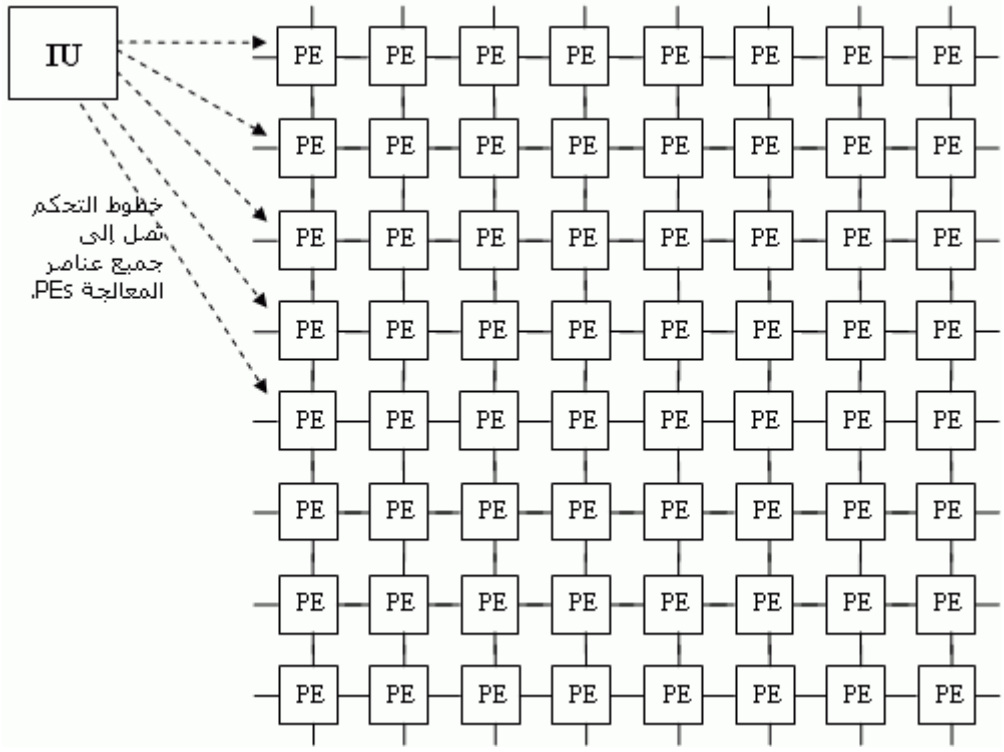


الشكل (2-3): SIMD

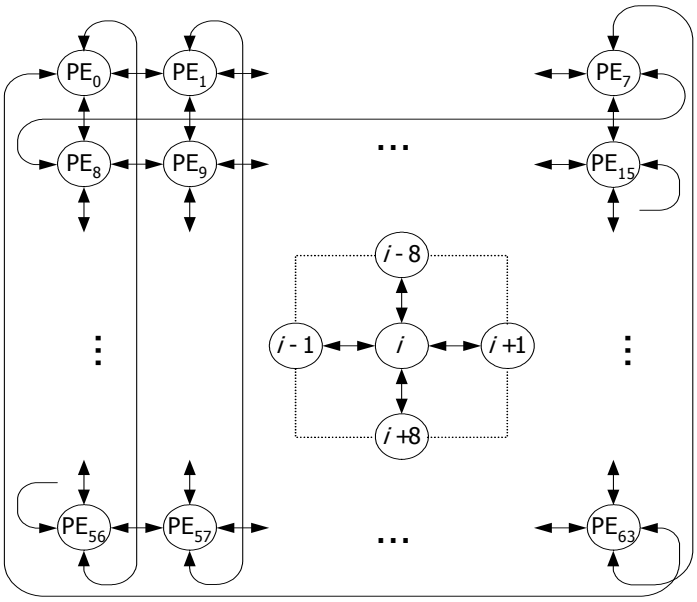
تستخدم في هذه الحاسبات آلاف المعالجات ، وتكون عادة متوسطة الأداء أو بسيطة جداً ، وجودتا وأدائها العالي ناتج عن العدد الكبير للمعالجات المستخدمة .

تحتل الحاسبات من النوع SIMD موقعاً بارزاً في تاريخ الحاسبات المتوازية، فأول حاسب متوازي تم تشييده كان من هذا النوع (وهو ILLIAC IV)، ولكن بسبب أن هذا النوع من الآلات يبنى باستخدام مكونات مخصصة لذا فإنه قل الإقبال عليها في السنوات القليلة الماضية.

يعتبر الحاسب (ILLIAC IV) مثالاً جيداً لتوضيح الآلات التي تنتمي إلى هذا النـوع . فهنالك وحدة تعليمات واحدة تصدر نفس الأمر إلى جميع عناصر المعالجة الأربع وسـتون. وكل عنصر معالجة له ذاكرة مكونة من ٢ كيلو من الكلمات (2K words) وذلك لتحميـل وتخزين ومعالجة المعطيات. وترتبط الأربع وستون عنصر معالجة معاً بشبكة ثنائية الأبعاد فيها ثمانية عناصر معالجة في كل جانب، ويمكن للعناصر المتجاورة إرسال واسـتقبال الرسـائل. وتلتف الارتباطات للمعالجات في الطرف العلوي لترتبط مع المعالجات في الطرف الـسفلي، وكذلك تلتف الارتباطات للمعالجات في الطرف الأيسر لترتبط مع المعالجـات في الطـرف الأيمن. (الالتفافات ظاهرة في الشكل(2-5)).

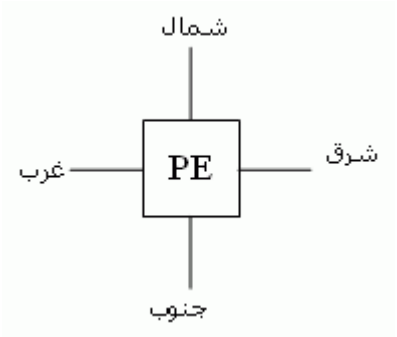


الشكل(4-2): يوضح الجهاز ILLIAC IV مع وحدة تعليمات واحدة (IU) و ٦٤ عنصر معالجة PEs



الشكل (5-2): يوضح طريقة التفاف الارتباطات بين المعالجات

وحيث أن كل عنصر معالجة (PE) يمكنه أن يرسل الرسائل في أربع اتجاهات فالروابط تعنون أو تعرف غالباً باتجاهات البوصلة (شمال، شرق، جنوب،غرب). ويدعى هذا أيضاً بشبكة اتصال NEWS. فمع تعليمة واحدة يمكن لعناصر المعالجة الأربع وستون أن تمرر الرسالة في اتجاه واحد، كالشمال مثلاً.



الشكل (6-2): يوضح العنونة باستخدام طريقة شبكة NEWS

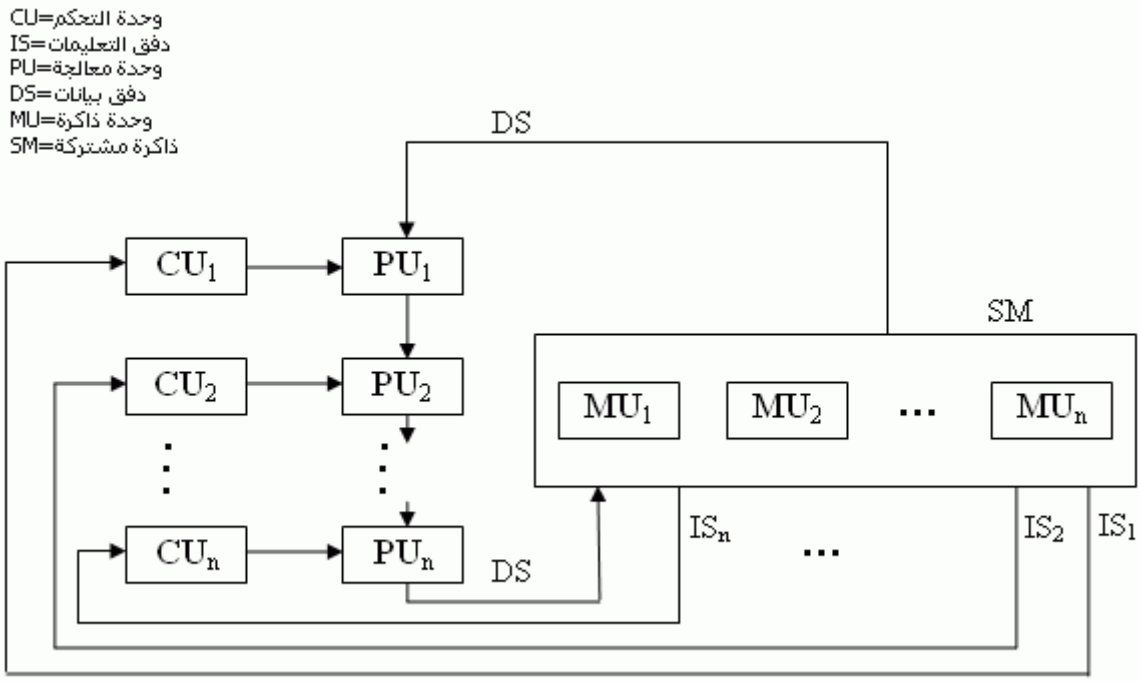
لم يصمم (ILLIAC IV) ليكون حاسباً عام الغرض، وإنما صمم لغرض خاص وهو حل المعادلات التفاضلية الجزئية. مثلاً يستخدم (ILLIAC IV) للتنبؤ بالطقس، والتنبؤ بـ الطقس

يستلزم بيانات كثيرة في فضاء ثلاثي الأبعاد. وفي هذه الحالة فالحل النموذجي هـو بتقـسيم الفضاء إلى ٦٤ قسماً، ووضع قسم واحد في كل معالج (PE). فالحلول للأربع وستين قسماً في هذه الطريقة تحسب بالتوازي. وعندما يحتاج عنصر المعالجة إلى بيانات من القسم الـاور فالاتصال يجب أن يبدأ على شبكة الربط.

من الآلات الهامة التي تتبـع لتـصنيف SIMD هـي: ILLIAC IV و ICL DAP و Goodyear MPP و MasPar MP-2.

2.1.3 الحاسبات متعددة تدفق التعليمات ووحيدة تدفق المعطيات MISD

حيث يتم في هذا النوع تنفيذ عدة تعليمات مختلفة على معامل واحد خلال الدورة الزمنية للحاسب. يتميز هذا النوع من الحاسبات بوجود عدد من المعالجات التي تعمل بشكل مستقل عن بعضها البعض. يتضمن كل معالج وحدة تحكم خاصة به تساعده على تنفيـذ المهـام الجزئية الموكلة إليه. يوجد القليل من الحاسبات المتوازية من نوع MISD وتكاد تكون أهميـة هذا النوع مقتصرة على تصنيف(Flynn) لأنه يتلاءم مع مبدأ التصنيف. وأن MISD هـي قليلة الاستعمال فهي تعتمد مبدأ العمل التسلسلي.



الشكل (2-7): MISD

2.1.4 الحاسبات متعددة تدفق التعليمات ومتعددة تدفق المعطيات MIMD

يتم في هذا النوع تنفيذ عدة تعليمات على معاملات مختلفة خلال الدورة الزمنية للحاسب . يتميز هذا النوع من الحاسبات بوجود عدد من المعالجات التي تعمل بشكل مستقل عن بعضها البعض. يتضمن كل معالج وحدة تحكم خاصة به تساعده على تنفيذ المهام الجزئية الموكلة إليه. هذه الحاسبات غير متزامنة وبالتالي فالحوادث التي تجري على معالج ما لا ترتبط بالحوادث التي تجري على المعالجات الأخرى . يمكن فرض نوع من التزامن بين هذه المعالجات فيما دعت الضرورات البرمجية لذلك ويتم هذا التزامن باستخدام بعض التعليمات الأولية المخصصة للتزامن أو عن طريق العتاد ويمكن أن يتم أيضاً عن طريق البرمجيات ونظم التشغيل . يوضح الشكل(2-8) أن كل معامل يعالج من قبل معالج ما بأمر ما

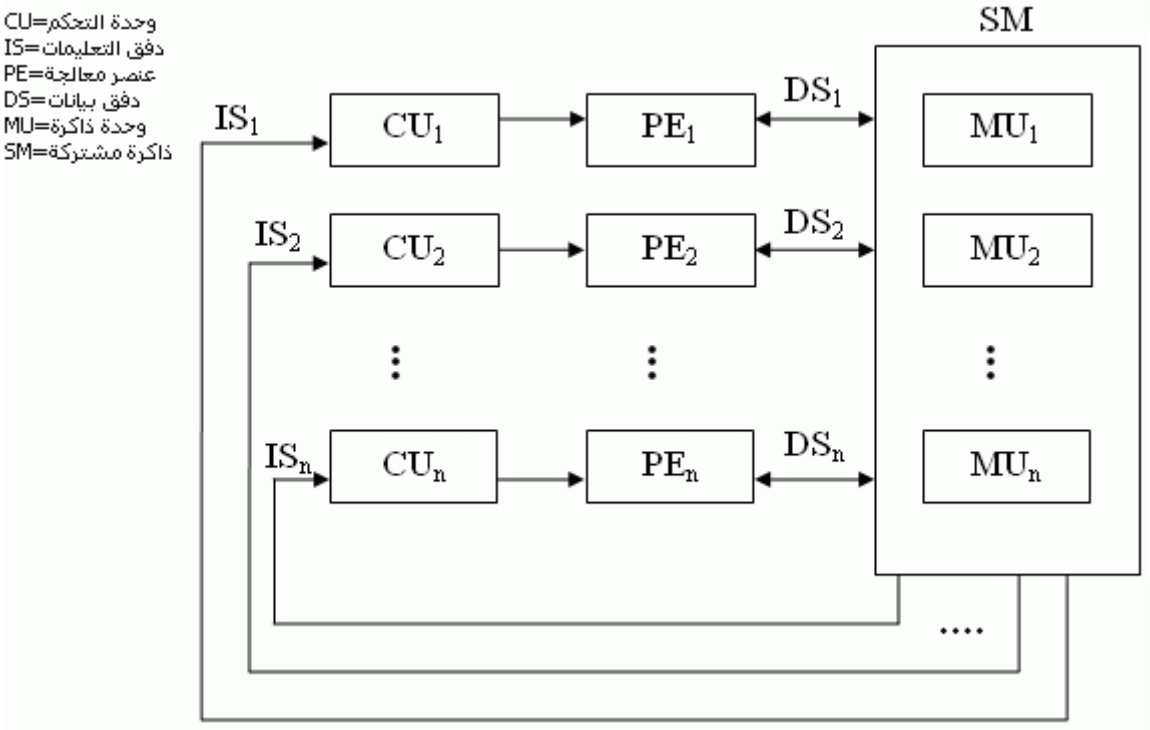
، وفي الوقت ذاته تعالج جميع المعاملات حسب كافة الأوامر ونتيجة هذه المعالجة تخـزن في الذاكرة. نلاحظ أنه عند المعالجة لا ينتظر أي معالج نتيجة من معالج آخر لأنه تعمـل جميـع المعالجات بشكل غير متزامن وعلى التوازي .

تستخدم المعالجة المتوازية في حاسبات **MIMD** على مستوى الخوارزميات حيث يقسم البرنامج إلى مهمات جزئية مستقلة جزئياً فيما بينها ، وتنفذ كل مهمة علـى معـالج مـن معالجات الحاسب . تتكون هذه الحاسبات من عشرات المعالجات وهي الحاسـبات الأكثـر عمومية في وقتنا الحاضر حيث يمكن استثمارها من أجل تطبيقات مختلفة ومتنوعة .

ويوجد لهذا التصنيف فئتين فرعيتين هامتين وهما:

(a) الذاكرة المشتركة (Shared memory).

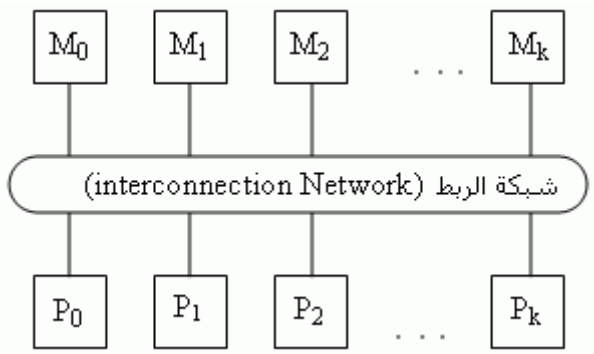
(b) تمرير الرسائل (message passing).



الشكل (2-8): MIMD

2.1.4-a الذاكرة المشتركة MIMD Shared Memory

في هذه الفئة الفرعية فإن أي معالج يتضمن وحدة تعليمات و وحدة حسابية تمكنه مـن أن يقرأ من أو يكتب في ذاكرة مشتركة.



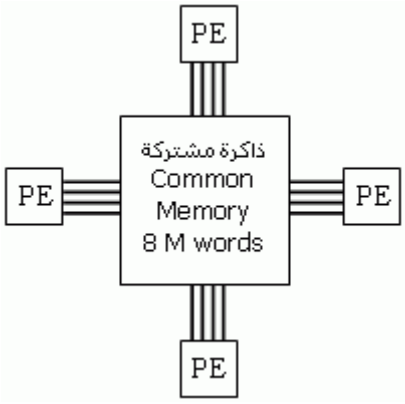
الشكل (2-9): نموذج الذاكرة المشتركة (MIMD Shared Memory)

في هذا النموذج ترتبط المعالجات مع وحدات الذاكرة بواسطة شبكة الربط، والتي يمكن أن تتخذ عدة أشكال تبعاً لنوع الآلة. شبكة الربط يمكن أن تثبت على شكل حلقة أو شبكة (Mesh).

يمكن لنا تشبيه هذا النموذج بلجنة تستخدم سبورة رئيسية مشتركة فيما بينـها لكـل الاتصالات، فأي عضو في اللجنة يمكن له أن يقرأ أي جزء من السبورة، ولكن يمكن لشخص واحد فقط أن يكتب على جزء معين من السبورة. ففي نموذج الذاكرة المشتركة هذا ربمـا يحصل تعارض أو تضارب في الذاكرة عندما يحاول معالجان الكتابة في نفس الجزء من الذاكرة في نفس الوقت، وأيضاً ربما تتداخل المعالجات مع بعضها عند الكتابة في نفس خلية الذاكرة المشتركة مما يتسبب في إيقاف وفشل العملية الحسابية. ولكي لا يصبح التداخل مشكلة يجب أن تزود الآلة بأقفال أو أي آلية للتزامن وذلك لضمان وجود معالج واحد فقط يتعامل مـع خلية الذاكرة المشتركة في الوقت الواحد.

وباختصار سوف نعرض لثلاث آلات تجارية تستخدم تقنية الذاكرة المشتركة، ويجب أن نأخذ في عين الاعتبار التنوع في شبكات الربط.

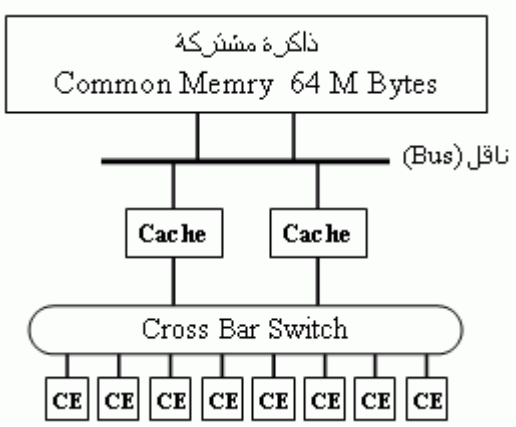
النموذج الأول: الحاسب الضخم Cray X-MP يحتوي على أربع معالجات، كل معالج له أربعة منافذ للذاكرة المشتركة يمكن أن تصل إلى ٦٤ مخزن ذاكرة (الشكل 10-2).



الشكل(10-2): يوضح الجهاز Cray X-MP/48 وكل معالج (PE) له أربع منافذ للذاكرة المشتركة

النموذج الثاني: The Alliant FX/8 minisupercomputer

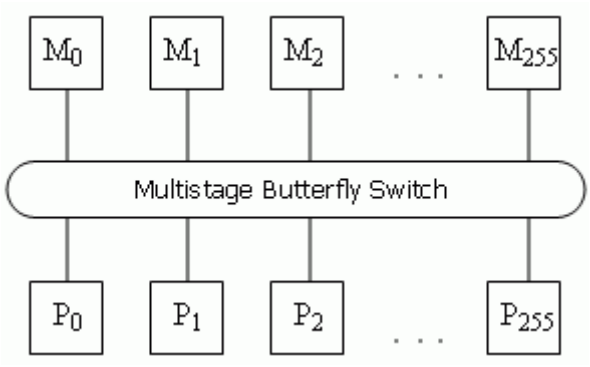
يحتوي هذا الحاسب على ثماني وحدات حسابية (CEs)، وتقتسم هذه الوحدات الحسابية فيما بينها ذاكرة مشتركة، وترتبط بطريقة Crossbar Switch إلى ذاكرتين خابيتين (cache) ذوات ٦٤ كيلو بايت. وتتعاقب الذاكرتان الخابيتان الولوج إلى الذاكرة المشتركة من خلال ناقل ذو سعة ١٨٨ ميغابايت/ثانية.



الشكل(11-2): يوضح الجهاز Alliant FX/8 بثمانية معالجات (CEs) تشترك في الذاكرة.

النموذج الثالث: The Bolt,Beranek and Newman (BBN)Butterfly

صنع هذا الحاسب في عام 1983، وهو حاسب متوازي عام الغرض، يتكون هذا الحاسب من 2 إلى 256 عقدة، وكل عقدة فيها معالج من نوع MC68000 يمكن أن تصل ذاكرة للمعالج إلى 4 ميغابايت، العقد ترتبط فيما بينها بطريقة شبكة الفراشة الملتفة (butterfly network).



الشكل (12-2): الجهاز BBN Butterfly يمكن فيه لأي عنصر معالجة P أن يتحول إلى أي وحدة ذاكرة M

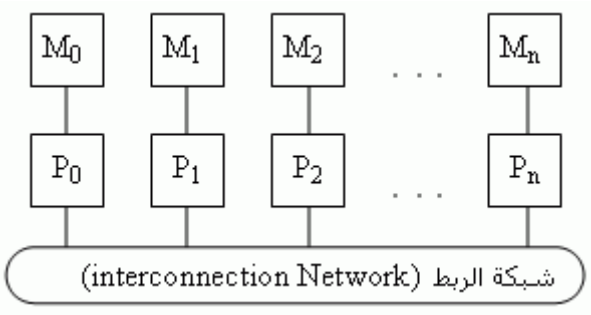
إن لكل آلات الذاكرة المشتركة MIMD سمات مشتركة. يزود نموذج الذاكرة المـشتركة مخزن للمبرمج . وهذا يتوافق مع وجهة النظر التقليدية لدى معظم المبرمجين . ولذلك استخدام نموذج الذاكرة المشتركة أسهل للبرنامج من نموذج تمرير الرسالة على سبيل المثال .

والعيب الواضح المشترك هو النقاط الساخنة (hot spot) في الذاكرة المـشتركة حيـث تحاول المعالجات أن تكتب على نفس خلية الذاكرة . و لأن المعالجات يجب أن تنتظر حـتى تكون خلية الذاكرة جاهزة فإن ذلك يمكن أن يعيق أداء النقاط الساخنة .

والعيب الأخر هو أن المبرمج والمترجم compiler ونظام التشغيل يجب أن تقرر كيـف يُقسّم البرنامج على عدة معالجات .

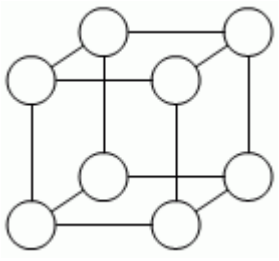
2.1.4-b تمرير الرسائل MIMD Message Passing

في نموذج تمرير الرسائل هذا فإن كل معالج له ذاكرة داخلية خاصة بـه، ولكـي تتواصـل المعالجات فيما بينها فإنا ترسل رسائل إلى كـل معـالج عـن طريـق شـبكة الـربط[1] (interconnection Network). (أنظر الشكل 2-13)



الشكل (2-13): يوضح نموذج تمرير الرسائل

وكما في نموذج الذاكرة المشتركة، فشبكة الربط يمكن أن تأخذ عدة أشكال مختلفـة. وشبكة الربط الشائعة في نموذج تمرير الرسائل هي "المكعب الثنائي متعدد الأبعاد" أو " nمن الأبعاد" (n-dimensional) ، بحيث أن كل بعد من الأبعاد يثبت عليه معالجين. مثلاً... في المكعب ثلاثي الأبعاد ستكون المعالجات في زوايا المكعب كما في الشكل(2-14):



الشكل (2-14): مكعب ثلاثي الأبعاد، المعالجات تتوضع على زواياه

[1] في الصفحات من ٤٤ وحتى ٥٢ شرح لشبكات الربط.

يتشابه نموذج تمرير الرسالة مع اللجنة فقط في حال إذا كان بمقدور الأعضاء أن يكتبوا ملاحظاتم لبعضهم . إن أسلوب توجيه الرسالة وسرعتها يشكل أمراً هاماً لأداء الآلات من هذا النوع .

ولهذا النموذج العديد من الحسنات، منها أنه لا يوجد ذاكرة مشتركة، وهـذا يعـني اختفاء المشاكل الناتجة عن التداخل والتضارب في الذاكرة والتي يعاني منها نموذج الـذاكرة المشتركة.كما أن النفاذ أو الوصول إلى الذاكرة سريع.

يخدم هذا النوع من الآلات غرضين هما:

الأول: يعمل كأداة اتصال لعبور قيم المعطيات بين المعالجات .

والثاني: يعمل كآلية تزامن للخوارزميات المتوازية.

وبما أن المعالجات لا تشترك في ذاكرة واحدة فإن نموذج تمرير الرسائل يمكن أن يحتوي على عدد كبير جداً من المعالجات، بخلاف نموذج الذاكرة المشتركة. فعلى سبيل المثـال... عندما نحاول إضافة معالجات أكثر إلى الجهاز CRAY X-MP فإن العدد الممكن للمنافذ إلى الذاكرة المشتركة محدود (الشكل (2-10)) ولا يمكن في هذه الطريقة زيادة عدد المعالجـات إلى أكثر من ١٦ معالج.

إن العيب الرئيسي في نموذج تمرير الرسائل هو الحمل الزائد الملقى على المبرمج، فلـيس كافياً أن يقوم المبرمج بتقسيم البرنامج على المعالجات، بل عليه أيضاً أن يوزع المعطيات. إن برمجة الآلات من نوع تمرير الرسائل تتطلب من المبرمجين أن يعيدوا النظر في الخوارزميات كي يكون استخدام الآلة أكثر كفاءة.

من الأمثلة لأجهزة تستخدم هذا النوع... Intel iPSC ولهـذه الآلـة 128 معـالج (128= 2^7 أي أن المكعب ذو 7 أبعاد). كذلك شركة nCUBE قامت بتسويق آلة لهـا 8192 معالج (8192=2^{13}، أي أن المكعب ذو 13 بعد).

وبالرغم من أن أكثـر الآلات التجاريـة تتبـع لأحـد أصـناف فلايـن (SISD, SIMD,MIMD) فإنه يوجد بعض التصاميم لا تتبع لتصنيف فلاين، مثلاً... بعض الآلات مثل الآلة ICL DAP يمكن أن توضع في أكثر من تصنيف. والبعض الآخر مثل dataflow و reduction لا تتبع لأي تصنيف من تصنيفات فلاين.

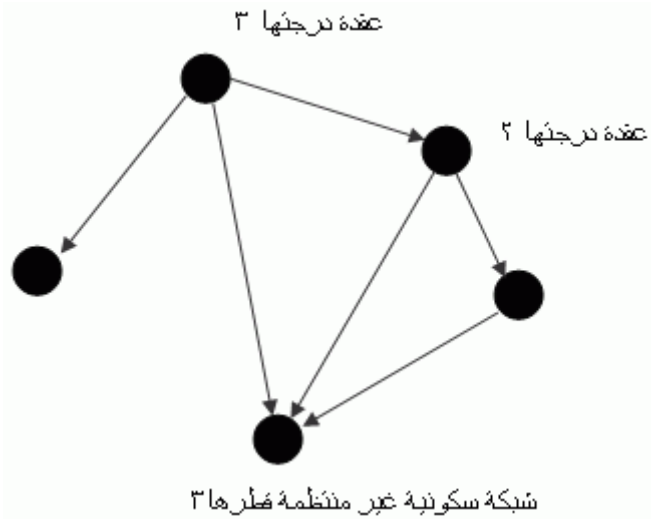
2.2 شبكات الربط (Interconnection Networks)

تعد شبكة الربط أداء الوصل بين المعالجات والذاكرة في الحاسبات المتوازية . و تتـألف عادة من خطوط اتصال وعناصر متخصصة لنقـل المعلومـات كالمبـدلات (switches) والموجهات (Routers). تختلف في بنيتها وفي استخدامها عن الشبكات الحاسوبية التي تصل عادة بين الحاسبات . فشبكة الربط في الحاسوب المتوازي تصل بين معالجـات متقاربـة و المسافة الفاصلة بينها صغيرة جداً، وقد تتواجد هذه المعالجات على البطاقـة نفـسها . أمـا الشبكات الحاسوبية فهي تربط حاسبات متباعدة تـصل المـسافة فيمـا بينـها إلى آلاف الكيلومترات في بعض الأحيان .

يعبر تراسل المعطيات عبر الشبكات الحاسوبية عن الولوج إلى حاسوب بعيـد جغرافيـاً بدف استخلاص بعض المعطيات أو لتنفيذ بعض البرامج الحاسوبية عن بعد. أمـا تراسـل المعطيات عبر شبكة الحاسوب المتوازي فيعبر عن التبعية الموجودة بـين مهمـات البرنـامج المتوازي وعن الارتباط بين المتحولات المستخدمة في المهمات المختلفة وبفضل شبكة الربط و تقنياتا التي تتطور بشكل مستمر فقد تمكن المصممون من بناء الحاسبات المتوازيـة ذات الذواكر الموزعة ، و تمكن شبكة الربط في هذه الحاسبات من تنفيذ مهمات البرنامج المتوازي رغم وجود التبعية فيما بينها . و تلعب تبولوجيا شبكة الربط ودرجة قرب المعالجـات مـن بعضها دوراً هاماً في برمجة هذه الحاسبات . و تطلق هذه الصفة أيضاً على الـشبكات الـتي تسمح بتغير طريق نقل المعطيات ديناميكياً بين معالجين ، و يتم هذا التغيير بدف إنشاء طرق جديدة لتراسل المعطيات ، و ترتبط هذه الخاصة أيضاً بالشبكات الديناميكية . وبالرغم مـن تنوع خاصية الشبكات إلا أن خاصية ثبوتية طرق نقل المعلومات تبقى أهمها .

2.2.1 الشبكات السكونية

تتميز الشبكات السكونية بتبولوجيا ثابتة . وتتمثل بمخطط بياني غير موجه عقده هي عقد معالجة الحاسوب المتوازي وخطوطه هي خطوط الاتصال . ولكل عقده في هـذه المخطـط درجة تتمثل بعدد خطوط اتصالها مع العقد الأخرى و تدل الدرجة على إمكانيـة تبـادل المعطيات بشكل مباشر بين العقدة والأخرى .



الشكل (2-15) : تمثل الشبكة السكونية بمخطط بياني

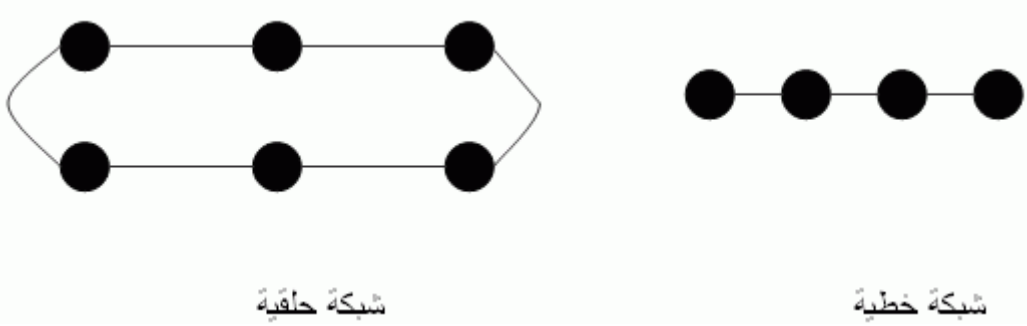
تكون الشبكة منتظمة Regular إذا كانت جميع عقدها من نفس الدرجة . و تتميز الشبكة السكونية أيضاً بقطرها وهو المسافة العظمى بين عقدتين من عقد الشبكة .

تقارن الشبكات السكونية فيما بينها بدلالة درجتها وقطرهـا ، وتفـضل الـشبكات ذات الدرجة الأعلى وذات القطر الأصغر فالدرجة العالية تدل على مرونة الاتصال بـين العقـد المختلفة مما يساهم في تسهيل برمجة هذه الحاسبات ، أما القطر الأصغر فيدل على مـسافات صغيرة بين العقد المختلفة و يترجم بزمن اتصال أصغر بين المعالجـات . أكثـر الـشبكات السكونية شهرة و استخداماً هي :

2.2.1.1 الشبكة الخطية والحلقية

و تتألف الشبكة الخطية من عدد من العقد تتصل فيما بينها بشكل خطي ، ويكون لكل عقدة عقدتين مجاورتين ما عدا العقدة الأولى والأخيرة حيث يكون لكل منهما عقدة مجاورة وحيدة . وقطر هذه الشبكة يتناسب مع عدد عقدها .

هذه الشبكة غير منظمة ويلجأ عادة لوصل طرفيها بحيث تصبح شبكة حلقيـة منتظمـة درجتها2 وقطرها يساوي إلى نصف عدد العقد ونظراً للتناظر تتميز شبكة الـربط بـسرعة نقلها للمعلومات ، وتؤثر خطوط الوصل وعناصر الاتصال تأثيراً مباشراً على هذه الـسرعة التي تقاس بعدد الخانات التي يمكن نقلها في الثانية M bits/s.

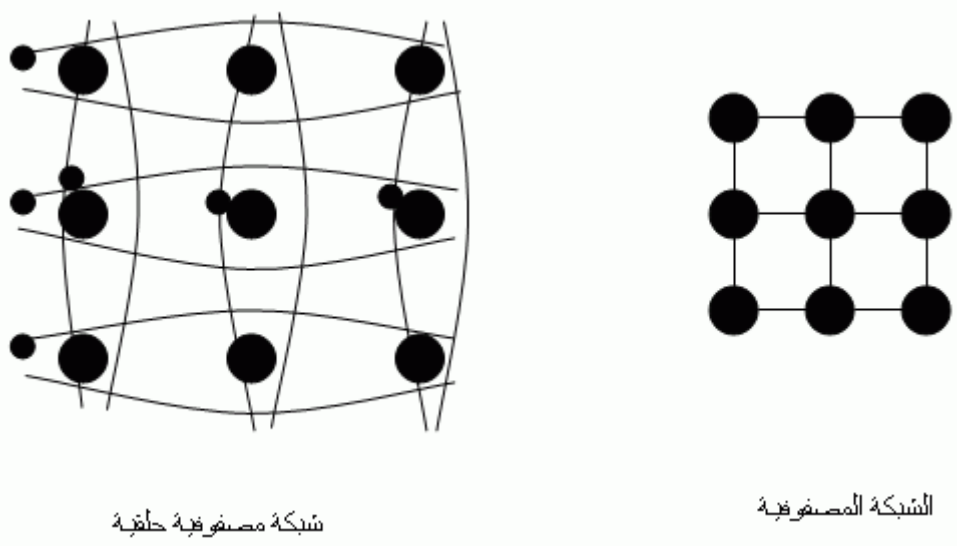


الشكل (2-16)

تستخدم في بعض الحاسبات معالجات خاصة من أجل إدراة وبرمجة عملية نقل المعطيات كما في عقدة حاسوب Paragon حيث يستخدم معالج i860 لإدارة الاتصال .

2.2.1.2 الشبكة المصفوفية و المصفوفية الحلقية

تكون هذه الشبكة على شكل مصفوفة لكل عقدة داخلية أربع عقد مجاورة ، أمـا العقـد الطرفية فدرجتها (2 or 3) . والشبكة المنتظمة المصفوفية الحلقية هي الأكثر استخداماً نظـراً للتناظر بين العقد من ناحية ولسهولة برمجتها من ناحية أخرى انظر الشكل (17-2).



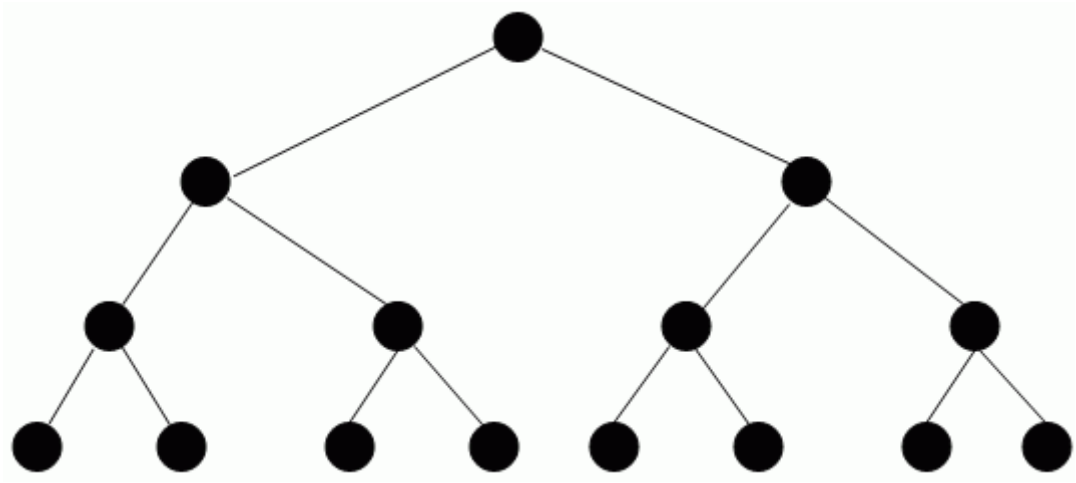
الشكل (17-2): الشبكة المصفوفية و المصفوفية الحلقية

تستخدم هذه الشبكات كثيراً في الحاسبات المتوازية الحالية نظراً لمرونة الاتـصال بـين العقد . و يعتبر معالج الترانسبيوتر من أكثر المعالجات ملاءمة لهذا النوع من التبولوجيا نظـراً لاحتوائه داخلياً على أربع قنوات اتصال .

تتلاءم هذه التبولوجيا مع خوارزميات معالجة الصور الرقمية التي تعتمد علـى حـساب الماورات وهي أكثر الشبكات استخداماً في هذا المال ورغم مرونة الاتصال بين المـاورات فإن التراسل بين العقد المتباعدة يحتاج إلى عمليات تسيير ولابد من إدارتا سواءً باسـتخدام المكتبات الخاصة أو ببرمجتها من قبل المستثمر .

2.2.1.3 الشبكات الشجرية

تتنوع أشكال هذه الشبكات تبعاً لعدد العقد المتصلة بكل عقدة أم وتختلف درجتـها وقطرها وفقاً لذلك انظر الشكل (2-18) . تتلائم هذه التبولوجيـا مـع البرمجـة المتوازيـة الديناميكية (عدد المهمات غير محدد مسبقاً) ومع البرامج من نمط السيد والخدم أو تلك التي تستثمر التوازي على عدة مستويات : مستوى الإجرائية و مستوى التعليمة مثلاً و يخـصص كل مستوى من مستويات الشجرة لمعالجة مستوى من التوازي . تستخدم هذه التبولوجيـا بالشكل المعقد كما في حاسوب CM-5 .

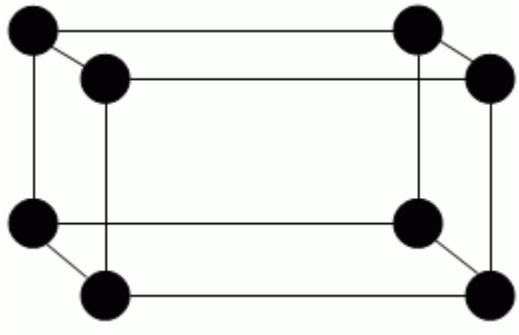


الشكل (2-18) : الشبكة الشجرية

2.2.1.4 الشبكات المكعبية

تتميز هذه الشبكات بطريقة بنائها التصاعدي و بدرجتها التي تساوي قطرها . تتـألف الشبكة المكعبية ذات الدرجة n من 2^n عقدة و يكون لكل عقدة n عقدة مجاورة و قطـر هذه الشبكة يساوي درجتها . فالشبكة المكعبية من درجة ١ تتألف من عقدتين متـصلتين والشبكات ذات الدرجة ٢ تتألف من ٤ عقد على شكل مربع ، الشكل (2-19) . يمكـن بناء الشبكة المكعبية ذات الدرجة (n) ابتداءً من شبكتين من الدرجة n-1 ، حيث يمكـن

بناء الشبكة المكعبية ذات الدرجة ٣ من شبكتين ذوات الدرجة ٢ . من أهم مميزات هـذه التبولوجيا هو قطرها الصغير نسبة إلى عدد عقدها .



الشكل(19-2) : شبكة مكعبية من الدرجة ٢

2.2.2 الشبكات الديناميكية

تتألف هذه الشبكات من عدد من عناصر الاتصال التي تسمح بمرور المعطيات باتجاهات مختلفة . يحدد طريق نقل المعلومات من معالج ما إلى معالج آخر عند تنفيذ عملية التراسـل و يؤخذ حمل الشبكة بعين الاعتبار عند تحديد هذا الطريق. تتكون هذه الـشبكات بـشكل رئيسي من مبدلات (Switchs) وموجهات (Router) . تتميز المبدلة بعدد مـن خطـوط الدخل و عدد مماثل من خطوط الخرج ، و تسمح بنقل المعلومات من أي خط دخل إلى أي خط خرج شرط أن يكون الخطان غير مشغولين بعملية نقل قيد التنفيذ .

أشهر الشبكات الديناميكية المستخدمة في الحاسبات المتوازية هي :

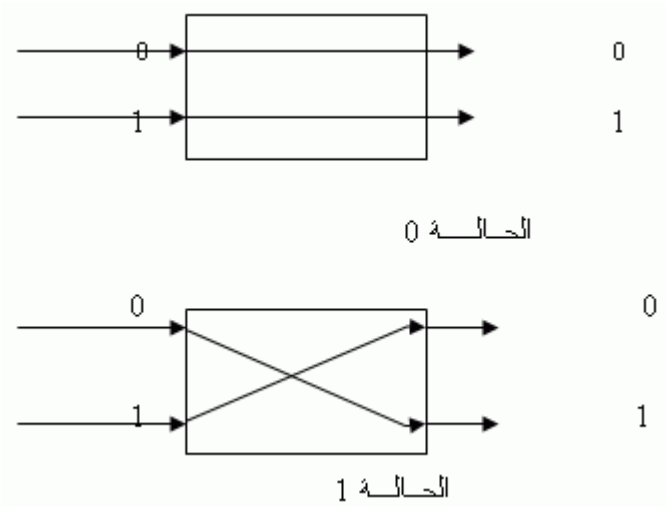
2.2.2.1 شبكة الناقل

يعتبر الناقل عنصر الاتصال المشترك و الوحيد بين المعالجات أو بين المعالجات من جهـة والذاكرة من جهة أخرى في هذه الشبكة . و للتنظيم المتشارك لابد من وجود عـدد مـن الخوارزميات التي تمكن من تنفيذ عمليات التراسل المختلفة بين المعالجات المتعددة . تعتمـد هذه الخوارزميات مبدأ الأفضلية حيث تحدد لكل معالج أفضلية تعتبر عنـد تحقيـق عمليـة

التراسل ، و تعتمد مبدأ FIFO(First In First Out) حيث يتم تنفيذ عمليات التراسل وفق ترتيب طلبها وتوجد خوارزميات أخرى تعتمد مبدأ الأفضلية المتغيرة .

2.2.2.2 مصفوفة المبدلات

تتألف هذه المصفوفة من عدد من المبدلات كل منها ذات مدخلين و مخرجين . و تتميز كل مبدلة بحالتين كما هو مبين في الشكل (20-2) :



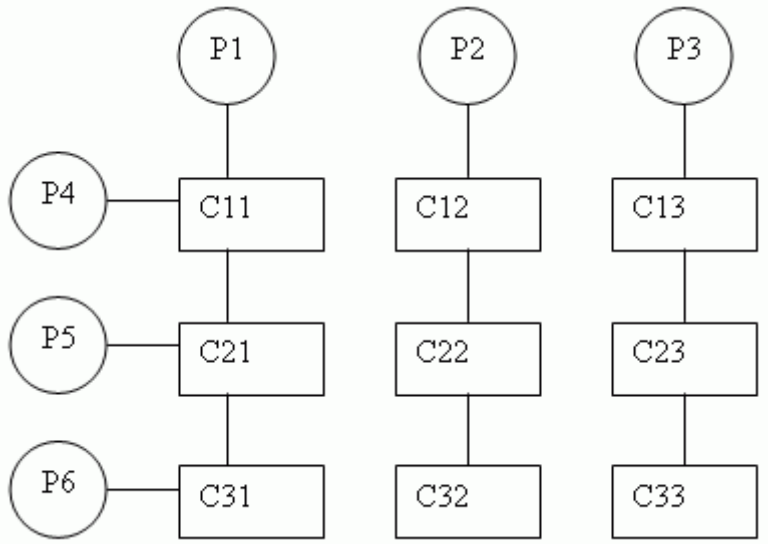
الشكل (20-2)

الحالة (0) : يتم فيها نقل المعلومات من المدخل (0) إلى المخرج (0) ومن المدخل (1)إلى المخرج (1).

الحالة (1) : يتم فيها نقل المعلومات من المدخل (1) إلى المخرج (0) ومن المدخل (0) إلى المخرج (1) .

يتم استخدام عدد من المبدلات من أجل نقل المعلومات بين معالجين من الشبكة ويحدد طريق نقل المعلومات عند تنفيذ عملية التراسل وفقاً لجاهزية المبدلات ويمكن استخدام كل مبدلة من أجل طريقي اتصال مختلفين شرط أن لا يشترك هذان الطريقان بالمداخل والمخارج. لا تؤمن هذه الشبكة الربط بين أي معالجين في أي لحظة . فهي شبكة مانعة تـؤخر تنفيـذ بعـض

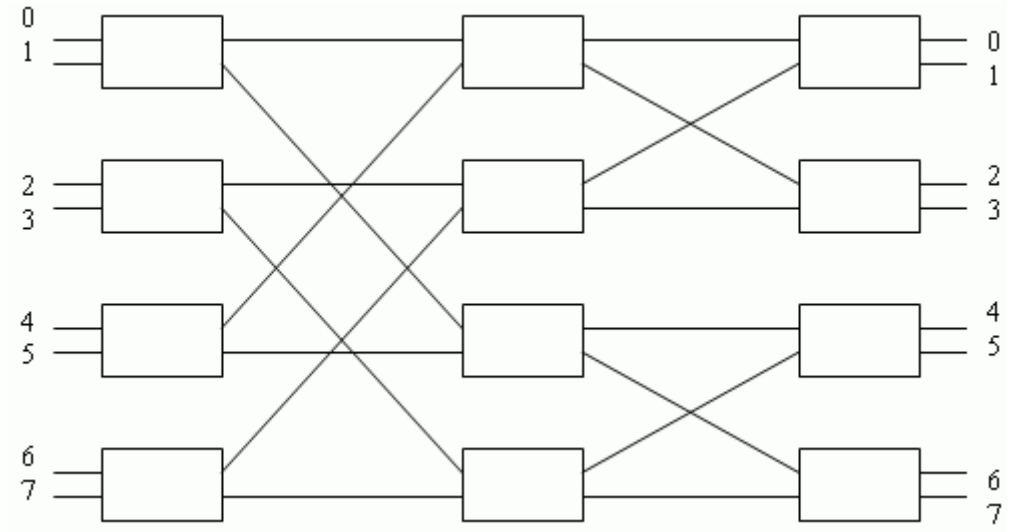
الاتصالات حتى تحرر بعض المبدلات . ومن الممكـن تصميم مصفوفة مبدلات غير ممانعـة ولكن حجم هذه الشبكة يكون كبيراً جداً وكذلك تكلفتها . تملك بعض المصفوفات خاصية قابلية إعادة التشكيل فتقوم الشبكة بتغيير بعض طرق الاتصال التي هي قيد التنفيذ من أجـل إنشاء طرق جديدة للاتصال . و طبعاً تخفف هذه الخاصية من تأثير الممانعـة. يـؤثر عـدد المبدلات التي تجتازها رسالة ما تأثيراً مباشراً على زمن تنفيذ عملية التراسل ولذا نسعى عادة إلى اختيار أقصر الطرق .



الشكل (21-2) : مصفوفة المبدلات

2.2.2.3 الشبكات متعددة الطبقات

تتألف هذه الشبكات من عدد من المبدلات وتقوم بشكل تصاعدي بإجراء عمليات ضـرب على الشبكات.



الشكل (22-2) : شبكة متعدد الطبقات

و تتألف كل شبكة من عدد من الطبقات مع عدد مداخل الشبكة ومخارجها . يوجد العديد من الطرق لبناء هذه الشبكات وتختلف الشبكات الناتجة بخواص عملية الاتصال .
أهم نتائج الدراسات الرياضية حول هذا النوع من الشبكات تلك التي تتعلق بالممانعة . فيبرهن رياضياً أنه لبناء شبكة غير ممانعة نحتاج إلى $n.\log_2(n)$ مبدلة وذلك باعتبار n عدد المداخل وعدد المخارج (شبكة لتأمين الاتصال بين n معالج) . فنحتاج إلى شبكة من ٢٤ مبدلة من أجل شبكة تصل بين ٨ معالجات ، انظر الشكل (22-2). يعتمد المصممون على شبكات أصغر حجماً وذات خواص مقبولة ببناء شبكات ذات عدد طبقات يساوي$\log_2(n)$ وبحيث يكون عدد المداخل وعدد المخارج مساوياً لعدد المعالجات . تستخدم أيضاً بعض الخوارزميات لإعادة تشكيل طرق الاتصال مما يخفف من الممانعة .

الفصل الثالث: مبادئ تصميم الخوارزميات المتوازية

يعتبر تطوير الخوارزميات عنصراً هاماً في حل المسائل باستخدام الحاسـبات الآليـة. وهناك نوعان من الخوارزميات؛ خوارزميات تسلسلية، وخوارزميـات متوازيـة، ويمكـن التعريف بشكل مبسط **للخوارزميات التسلسلية** بأنها وصف (أو تسلسل مـن الخطـوات الأولية) لحل مسألة معطاة على حـاسب تسـلسلي، أما **الخوارزميات المتوازية** (Parallel Algorithms) فهي تصف كيفية الحل لمسألة معطاة باستخدام عدة معالجـات. إن تـصميم الخوارزميات المتوازية يتطلب أكثر من مجرد تحديد خطوات الحـل، فعلـى أقـل تقـدير، للخوارزميات المتوازية بعد إضافي للتزامن، وعلى مصمم الخوارزميّة أن يحدد مجموعـة مـن الخطوات يمكن أن يتم تنفيذها سوياً بنفس الوقت. وهذا التحديد ضروري للحصول على أي زيادة في الأداء من استعمال الحاسب المتوازي. يمكن تحديد الميـزات الـتي تتـصف بـها الخوارزميات المتوازية أو في بعض منها على النحو الآتي:

- تحديد أجزاء العمل الذي يمكن أن يؤدى بشكل متزامن.
- توضيع (أي إسناد) الأجزاء المتزامنة مـن العمـل في عـدة إجرائيـات(أو معالجات) تعمل بالتوازي.
- توزيع المعطيات المدخلة والمخرجة والمعطيات الوسيطة المرتبطة بالبرنامج.
- إدارة عملية الوصول إلى المعطيات المشتركة بين عدة إجرائيات.
- مزامنة المعالجات في المراحل المختلفة لتشغيل البرنامج المتوازي.

3.1 مفاهيم أساسية

• **تصميم الخوارزميات المتوازية:** هناك خطوتان رئيسيتان مستخدمتان في تصميم الخوارزميات المتوازية: **الأولى-** تقسيم العملية الحسابية إلى عدة عمليات حسابية صغيرة، **والثانية-** وضع هذه العمليات المقسمة في عدة معالجات لكي تنفذ بشكل متوازٍ. وسوف نعرض لاحقاً هاتين الخطوتين المستخدمتين في تصميم الخوارزميات المتوازية من خلال المثالين: ضرب مصفوفة بشعاع، وعملية استعلام في قواعد البيانات، إضافة إلى التعريف بعدد من المفاهيم المستخدمة.

• **التقسيم(Decomposition):** هو عملية تفكيك العملية الحسابية إلى أجزاء أصغر.

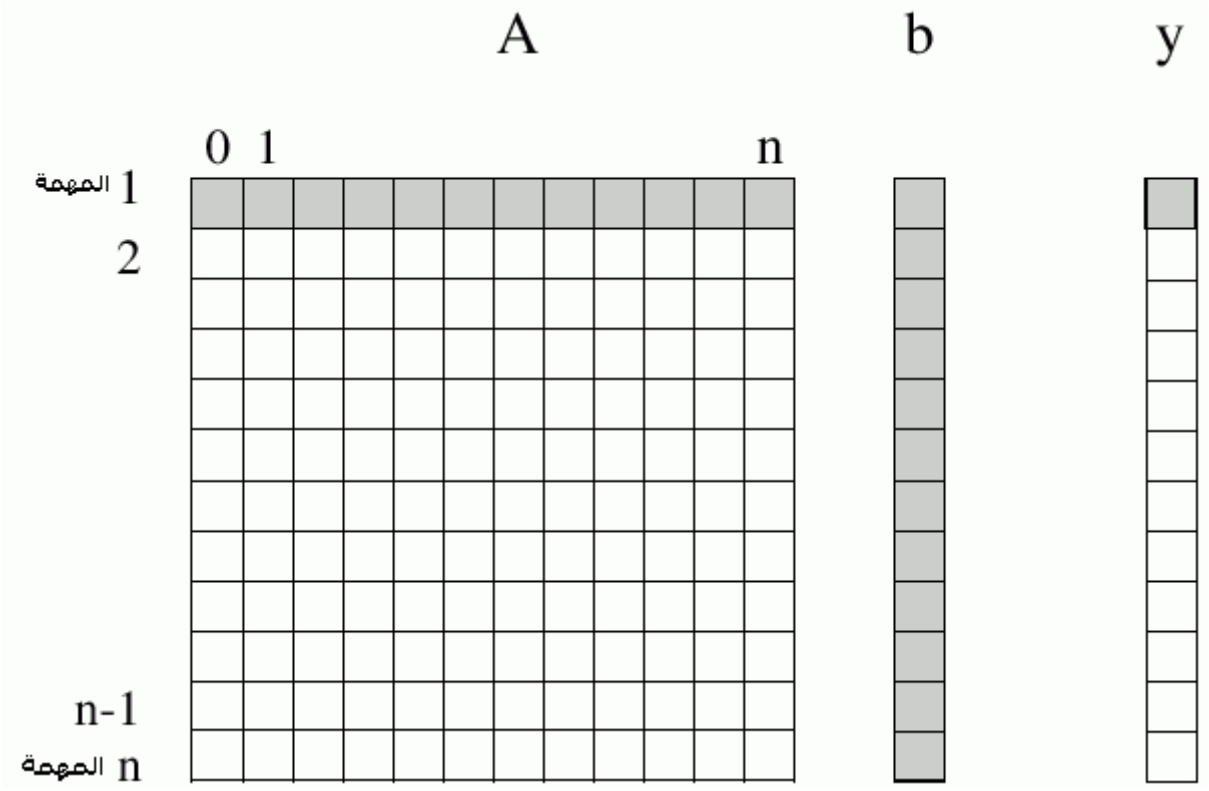
• **المهام** (Tasks): هي وحدات من العملية الحسابية معرفة بواسطة المبرمج والتي تمثل أجزاء العملية الحسابية الرئيسية التي تم الحصول عليها بواسطة التقسيم. والتنفيذ **المتزامن (Concurrency)**(أي بنفس الوقت) للمهام المتعددة هو المفتاح الأساسي لإقلال الزمن اللازم لحل المسألة بكاملها. وقد تكون المهام -للمسألة المقسمة- ليست كلها من نفس الحجم.

• **مخططات التبعية** (Depenency Graph): هي الرسم التجريدي الذي يستخدم للتعبير عن الاعتمادية أو التبعية فيما بين المهام و الترتيب/النظام النسبي للتنفيذ.

مثال(1-3): ضرب مصفوفة بشعاع

باعتبار أننا نريد إجراء عملية الضرب على المصفوفة A بحجم n×n مع الشعاع b، فسينتج لدينا شعاع آخر y. إن حاصل عملية الضرب y[i] يكون ناتجاً عن ضرب السطر i من A مع كامل الشعاع b. وللإيضاح فإن: $y[i] = \sum_{j=1}^{n} A[i, j].b[j]$ وكما هو موضح في الشكل(1-3) فإن عملية حساب كل قيمة y[i] يمكن أن تعتبر كمهمة.

كما يمكن تقسيم هذه المسألة إلى عدة مهام كما في الشكل **(3-4)** حيث قسمت المسألة إلى ٤ مهام، بحيث أن كل مهمة تقوم بحساب n/4 من الناتج.



الشكل(1-3): مسألة ضرب مصفوفة بشعاع مقسمة إلى n مهمة، حيث n هي عدد أسطر المصفوفة. الجزء الذي تتعامل معه(مدخلات ومخرجات) للمهمة ١ موضح باللون الغامق.

نلاحظ بأن جميع المهام المحددة في الشكل**(3-1)** هي مهام مستقلة ويمكن تنفيذها سوياً أو على أي تسلسل. وبشكل عام، في بعض المسائل قد تكون بعض المهام فيهـا بحاجـة إلى بيانات ناتجة عن مهام أخرى ولذا فإن عليها الانتظار إلى أن تُنهي هذه المهام أعمالها.

وفي مخطط التبعية تعتبر **العقد** كمهام، أما الخطوط التي تصل بين العقد(يطلـق عليهـا **أضلاع**) فتدل على الاعتمادية بين المهام. فالمهمة التي تتطابق مع أحد العقد لا يمكن تنفيذها

إلا حين انتهاء تنفيذ جميع المهام التي تدخل إليها(أي أن الخط الواصل يكون داخل للمهمـة وليس خارجا منها).

مثال(2-3) إجرائية الاستعلام من قواعد البيانات

يوجد في الجدول **(3-1)** عرض لقاعدة بيانات علائقية خاصة بسيارات، وكل صف في هذا الجدول هو سجل يحتوي على بيانات عن سيارة محددة، مثل المَعرف ID، وسنة الإنتاج year، واللون color، الخ..

ID#	Model	Year	Color	Price
4523	Civic	2003	Blue	55,000
3476	Corolla	1999	White	45,000
7623	Camry	2003	Green	59,500
9834	Prius	2001	Green	48,000
6734	Civic	2001	White	47,000
5342	Altima	2001	Green	49,000
3845	Maxima	2001	Blue	52,000
8354	Accord	2000	Green	48,000
4395	Civic	2001	Red	47,000
7352	Civic	2002	Red	48,000

الجدول (3-1): قاعدة بيانات لتخزين معلومات عن السيارات.

لنفترض أننا نريد إجراء الاستعلام التالي:

```
MODEL="Civic" AND YEAR="2001" AND (COLOR="Green" OR
COLOR="White")
```

يقوم هذا الاستعلام بالبحث عن جميع السيارات التي من النوع Civic والتي أنتجـت في السنة ٢٠٠١ ولها أحد اللونين: الأخضر أو الأبيض.

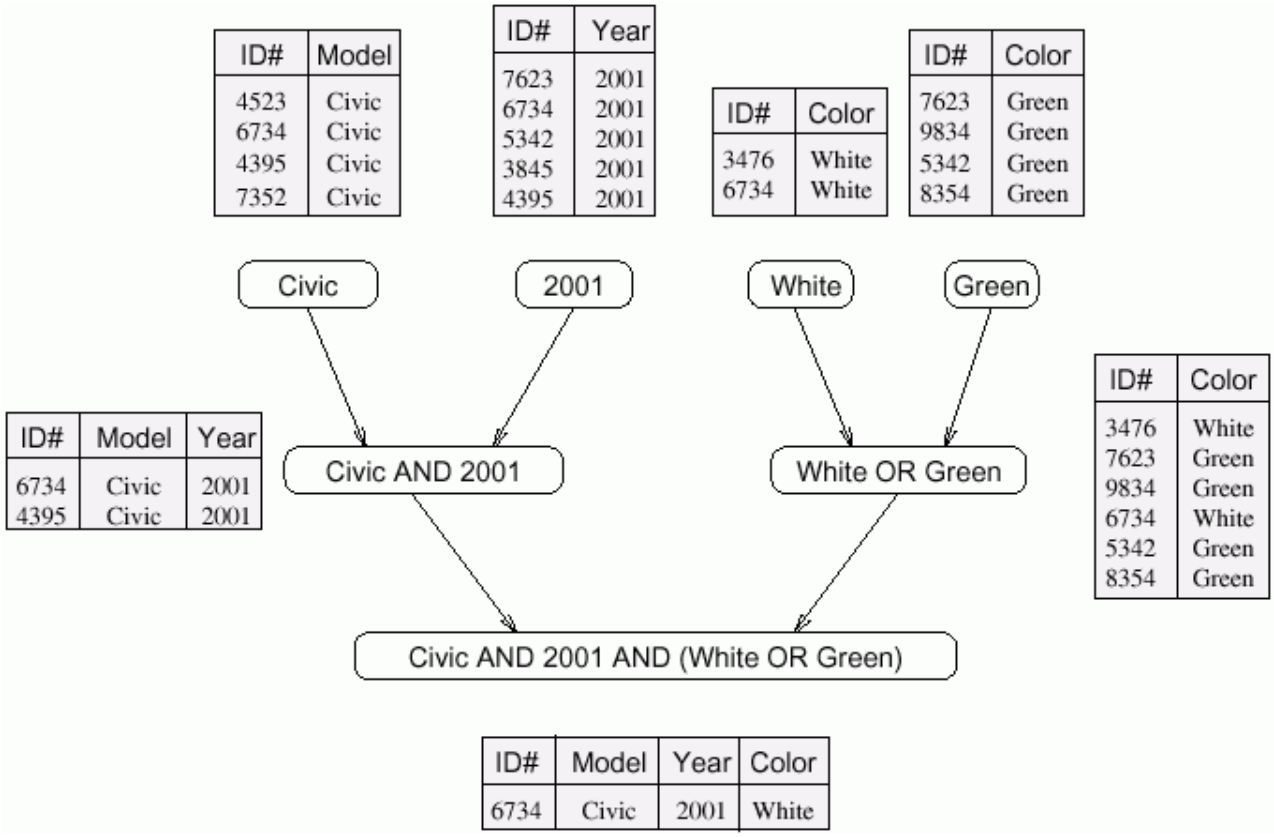
في قواعد البيانات العلائقية (Relational Database)، يتم هذا الاستعلام بواسطة إنشاء عدد من الجداول الوسيطة، وإحدى الطرق الممكنة لهذه المسألة هي إنشاء الجداول الأربعة التالية:

⇐ جدول يحتوي على جميع السيارات من نوع Civic.
⇐ جدول يحتوي على جميع السيارات التي أنتجت في عام ٢٠٠١.
⇐ جدول يحتوي على جميع السيارات ذات اللون الأخضر.
⇐ جدول يحتوي على جميع السيارات ذات اللون الأبيض.

ثم بعد ذلك تتم العملية بواسطة دمج هذه الجداول عن طريق حساب التقاطعـات أو الاتحادات بين الجداول زوجاً زوجاً. وعلى وجه التحديد سيتم حساب التقاطع للجـدولين "السيارات من نوع Civic" و "السيارات التي أنتجت عام ٢٠٠١" وذلك لإنشاء جـدول يحتوي على سيارات Civic التي أنتجت عام ٢٠٠١.

وبنفس الأسلوب، سيتم إجراء اتحاد لجدولي "اللون الأخضر" و "اللون الأبيض" وذلـك لكي يتم إنشاء جدول لجميع السيارات ذات اللون الأخضر أو الأبيض. وفي النهاية سـيتم إجراء التقاطع للجدول الذي يحتوي على سيارات Civic 2001 مع الجدول الذي يحتوي على جميع السيارات الخضراء أو البيضاء اللون، وبذلك يتم الحصول على نتيجة الاستعلام.

يمكن للحسابات المختلفة التي استخدمت لمعالجة الاستعلام في المثال الـسابق أن تمثـل بواسطة مخطط التبعية الموضح في الشكل**(2-3)**.



الشكل(2-3): الجداول المختلفة والعلاقة بينها في عملية الاستعلام.

كل عقدة في الشكل تمثل مهمة تتطابق مع جدول وسيط بحاجة إلى أن يُحسب، أما الأسهم التي بين العقد فتوضح العلاقة (أو التبعية) بين المهام. فعلى سبيل المثال، قبل أن نتمكن من حساب الجدول الذي يحتوي على Civic 2001 يجب أولا أن نقوم بحساب الجدولين "سيارات Civic" و "سيارات ٢٠٠١".

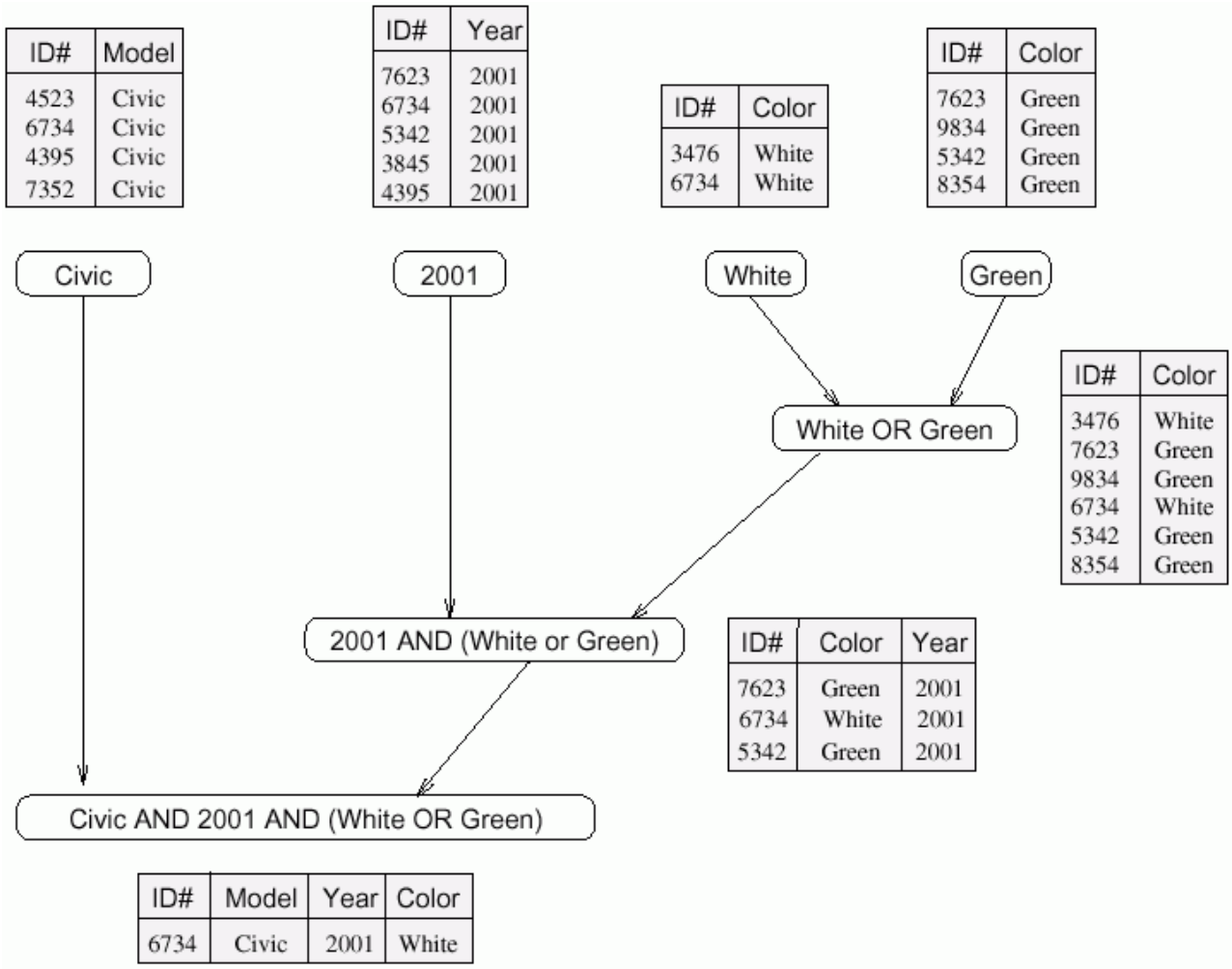
يوجد عدة طرق للحصول على بعض الحسابات، وخصوصا تلك التي تستخدم المعاملات، مثل: الجمع ، الضرب أو AND و OR المنطقيتان. فالطرق المختلفة لترتيب الحسابات تؤدي إلى مخططات تبعية مختلفة وذات مزايا مختلفة أيضاً. وللدلالة على ذلك، فاستعلام قاعدة البيانات الوارد في المثال(2-3) يمكن أن يتم حله بالأسلوب الآتي:

أولا: تحديد جدول يحتوي على السيارات ذات اللون الأخضر أو الأبيض.

ثانيا: إجراء تقاطع لجدول "السيارات ذات اللون الأخضر أو الأبيض" مـع الجـدول: "سيارات أنتجت في عام ٢٠٠١"

ثالثا: تدمج النتائج مع جدول "سيارات Civic".

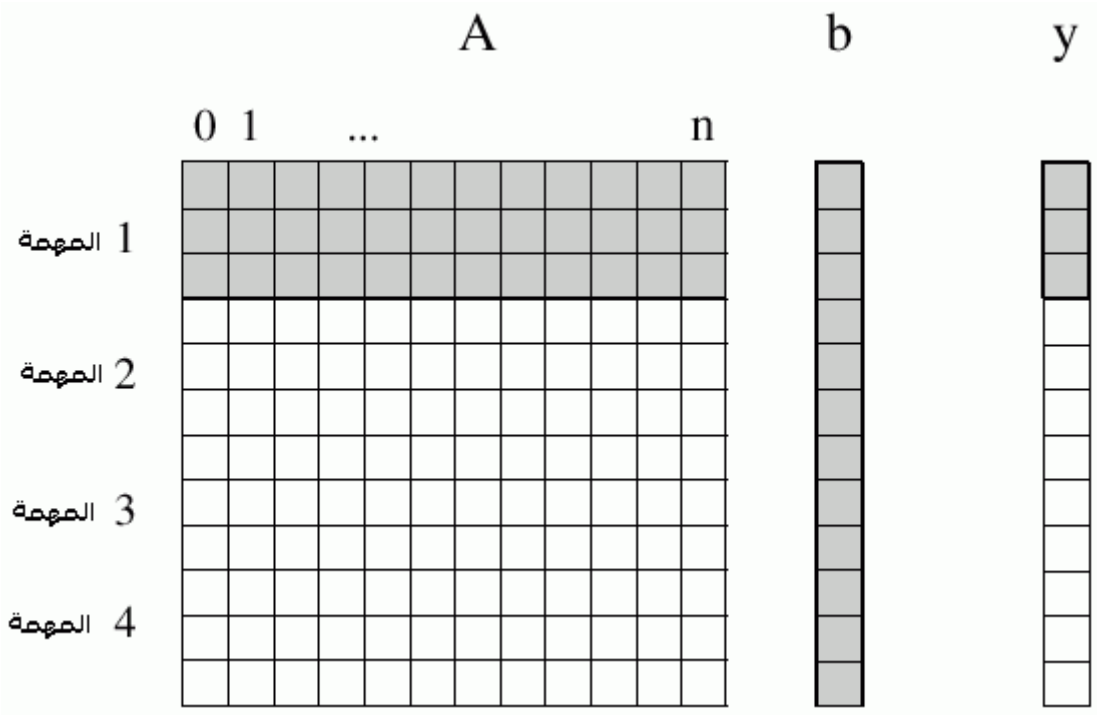
يوضح الشكل(3-3) هذه الخطوات من خلال مخطط التبعية:



الشكل(3-3): مخطط التبعية لعملية الاستعلام.

• **الحبوبية**(Granularity): يطلق على حجم وعدد المهام في المسألة الجزأة التقسيم الحبوبي. والتقسيم إلى عدد كبير من المهام الصغيرة يطلق عليه **الحبوبية الناعمة.** أما التقسيم إلى عدد صغير من المهام الكبيرة فيطلق عليه **الحبوبية الخشنة.**

على سبيل المثال، التقسيم لمسألة عملية ضرب مصفوفة بشعاع الواردة في المثال(1-3) تعتبر تقسيماً حبوبياً ناعماً وذلك لأن كل مهمة من المهام الكثيرة تقوم بتنفيذ عملية الضرب لسطر واحد. أما في الشكل(4-3) ففيه عرض لتقسيم من نوع الحبوبية الخشنة لنفس المسألة إلى ٤ مهام، بحيث تقوم كل مهمة بتنفيذ n/4 من العمل لكامل الشعاع الناتج.



الشكل(4-3): مسألة ضرب مصفوفة بشعاع مقسمة إلى أربعة مهام. الجزء الذي تتعامل معه(مدخلات ومخرجات) للمهمة ١ موضح باللون الغامق.

يتعلق مفهوم التقسيم الحبوبي **بدرجة التزامن.** فالحد الأقصى لعدد المهام التي يمكن تنفيذها بشكل آني في برنامج متوازي في أي وقت معطى يطلق عليه **"الدرجة العظمى**

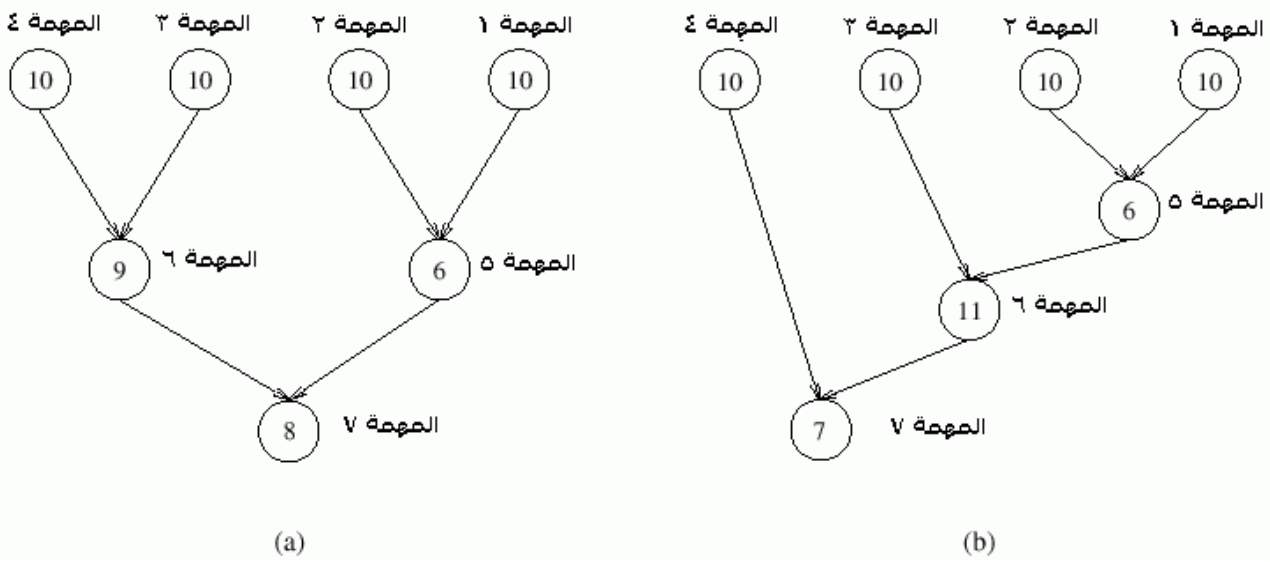
للتزامن". وفي أغلب الحالات، تكون الدرجة القصوى للتزامن أقل من عدد المهـام الكلـي وذلك بسبب العلاقة فيما بين المهام. فعلى سبيل المثال، الدرجة القصوى للتزامن في مخطـط التبعية الموضح في الشكل **(3-2)** و شكل **(3-3)** هو 4. وفي مخططات التبعية هذه، تحـصل الدرجة القصوى للتزامن في البداية عند ما يتم حساب الجداول الأربعة(النوع،السنة،اللـون الأبيض،واللون الأخضر) بنفس الوقت.

وبشكل عام، درجة التزامن العظمى لمخططات التبعية الشجرية يساوي دائمـا لعـدد التفرعات في الشجرة.

هناك مؤشر هام للدلالة على أداء البرامج المتوازية، هذا المؤشر هو "**متوسـط درجـة التزامن**"، والذي يمكن حسابه بأخذ المتوسط لعدد المهام التي يمكن تنفيذها تزامنيا خـلال كامل مدة تشغيل البرنامج. ويمكن أن يزداد كل من المعدل والحد الأقصى لدرجة التـوازي كلما كان التقسيم الحبوبي للمهام أصغر (أنعم). فعلى سبيل المثال التقسيم لمـسألة ضـرب مصفوفة بشعاع الوارد في الشكل**(3-1)** له تقسيم حبوبي صغير ودرجة تزامن عاليـة. أمـا التقسيم لنفس المسألة في الشكل**(3-4)** له تقسيم حبوبي كبير ودرجة تزامن منخفضة.

تعتمد درجة التزامن أيضا على شكل مخطط التبعية والتقسيم الحبوبي ذاته. وبشكل عام، ليس هناك ضمان لتماثلهما في درجة التزامن. فعلى سبيل المثال يعتبر الشكل **(3-5)** تجريـدا لمخططي التبعية في الشكلين (**(3-3)** و **(3-2)**) على التوالي، والعدد المكتوب بـداخل كـل عقدة يمثل كمية العمل المطلوب لإكمال المهمة المقابلة لهذه العقدة.

إن معدل درجة التزامن لمخطط التبعية الموضـح في الـشكل **(3-5.a)** هـو **2.33**، وفي الشكل **(3-5**.b**)**هو 1.88، مع أن كلا المخططان يعتمدان نفس التقسيم.



الشكل(5-3): تجريد لمخططي التبعية للشكلين (3-3) و(3-2)

من مزايا مخطط التبعية أنه يحدد معدل درجة التزامن لأي تقسيم حبوبي معطـى عـن طريق **المسار الحرج**، سنشير للعقد التي ليس لها أضلاع داخلة إليها بـعقد البداية، أما العقد التي لا يخرج منها أضلاع فسنشير لها بـعقد النهاية. وعلى هذا **فالمسار الحرج** هو أطـول خط يصل بين أي زوجين من عقد النهاية والبداية. وأما الموع لكمية العمل للعقد الواقعة على المسار الحرج يعرف بطول المسار الحرج، بحيث أن كمية العقدة هي كمية العمل للمهمة المطابقة لهذه العقدة. أما نسبة إجمالي كمية العمل للمسار الحرج فُتعـرَف بـ**معـدل درجـة التزامن**. ولذلك فالمسار الحرج الأقصر يؤدي إلى درجة تزامن أعلى. فعلى سبيل المثال، طول المسار الحرج لمخطط التبعية الموضح في الشكل **(3-5.a)** هـو 27، أمـا للـشكل **(3-5.b)** فالطول هو 34، ونظرا لأن مجموع كمية العمل اللازمة لحل المسألة باستخدام أسلوبي التقسيم هو 63 و 64 على التوالي فإن معدل درجة التزامن لمخططي التبعية هـو 2.33 و 1.88 علـى التوالي.

على الرغم من أنه قد يظهر للبعض بأنه يمكن إنقاص الزمن اللازم لحل المسألة باستخدام تقسيم ذو حبوبية عالية، إلا أن هذه الحالة لا تصادف كثيراً في الأحوال العملية. ففي العادة يكون هناك حد أعلى لكمية التقسيم الحبوبي الناعم التي تسمح بها المسألة، فمثلا هناك N^2 عملية ضرب ومثلها للجمع في مسألة ضرب مصفوفة بشعاع التي وردت في المثال **(3.1)** فهذه المسألة لا يمكن تقسيمها لأكثر من $O(N^2)$ مهمة حتى ولو استخدامنا أكثر أنواع التقسيم نعومة في التحبيب.

• **تفاعل المهمة**(Task-Interaction): يعتبر عامل عملي آخر هام يقلل من قدرتنا على تحقيق التسريع غير المحدود(نسبة زمن التنفيذ التسلسلي إلى المتوازي) من جراء استعمال التوازي. هذا العامل هو التفاعل بين المهام التي تعمل على معالجات مختلفة. فالمهام المقسمة في المسألة تتشارك فيما بينها مدخلات ومخرجات وبيانات وسيطة. والتبعية في مخطط التبعية تنتج غالباً من حقيقة مخرجات إحدى المهام لتكون مدخلات لمهام أخرى. فعلى سبيل المثال في مثال استعلام قاعدة البيانات، تتشارك المهام فيما بينها بالبيانات الوسيطة؛ فالجدول المنشأ بواسطة إحدى المهام يستخدم من قِبَل مهمة أخرى كمدخلات. واعتمادا على التعريف للمهام و نموذج البرمجة المتوازية فقد يحصل هناك تفاعل فيما بين المهام التي تظهر مستقلة في مخطط التبعية. فمثلا.. في التقسيم لمسألة ضرب مصفوفة بشعاع، وعلى الرغم من أن جميع المهام مستقلة عن بعضها البعض، فإن جميع هذه المهام تتطلب الوصول إلى كامل الشعاع b. ونظراً لأن هناك نسخة واحدة من الشعاع b، فعلى المهام أن تُرسل وتستقبل الرسائل من الجميع لكي تصل إلى الشعاع b في نموذج الذاكرة المشتركة.

3.2 الإجرائيات والمقابلة

إن الإجرائيات هي أدوات حسابية منطقية(Logical) تقـوم بتنفيـذ المهـام. أمـا المعالجات فهي وحدات عتادية (Hardware) والتي تقوم بتنفيذ العمليات الحسابية فيزيائيـاً. وفي الغالب فإننا عندما نذكر مصطلح الإجرائيات فهناك تطابق بينه وبين مصطلح المعالجات، وعموماً فإنه يمكن توضيع أكثر من إجرائية على المعالج الواحد.

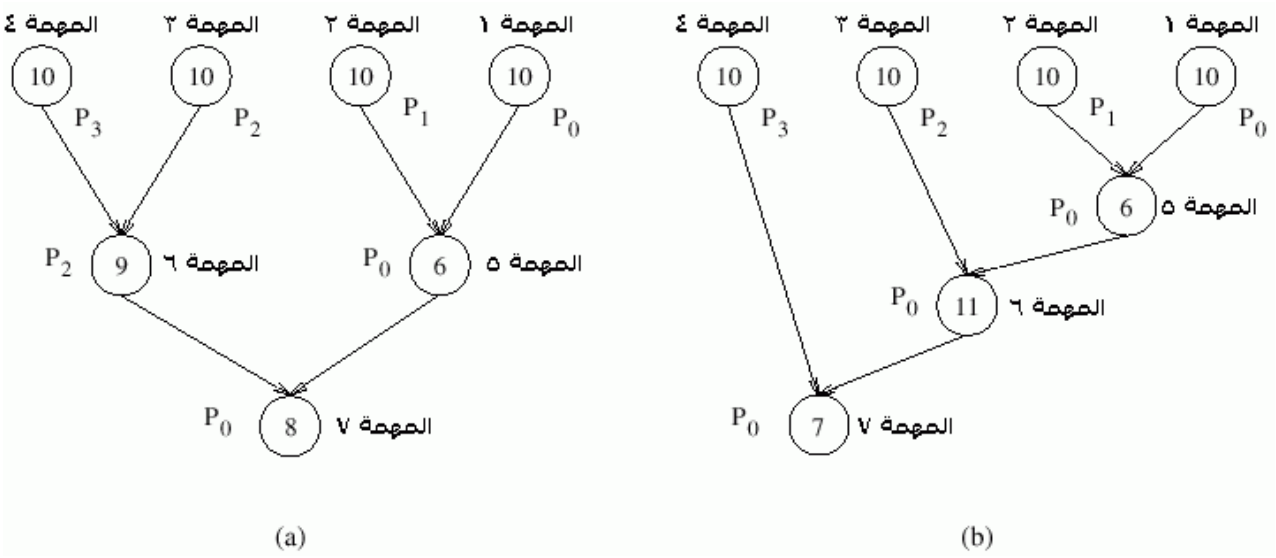
لكي نحصل على تسريع أعلى من التنفيذ التسلسلي يجب على البرنـامج المتـوازي أن يكون لديه عدة إجرائيات تعالج عدة مهام بنفس الوقت، وتسمى الآلية المستخدمة لتوزيـع المهام لتنفيذها على عدة إجرائيات بالمقابلة (mapping).

فمثلاً: في المثال**(4-3)** في ضرب المصفوفات يمكن أن تخصص أربع إجرائيـات لمهمـة حساب مصفوفة جزئيه من c (سيذكر المثال لاحقاً).

يلعب مخططي التبعية و التفاعل بين المهـام دوراً هامـاً في اختيـار المقابلـة الجيـدة للخوارزميات المتوازية. والمقابلة الجيدة يجب أن تسعى إلى زيادة استعمال التـوازي وذلـك بمقابلة المهام المستقلة على إجرائيات مختلفة، أيضا لابد أن تسعى إلى تقليص الـزمن الكلـي وذلك بضمان أن الإجرائيات جاهزة لتنفيذ المهام على المسار الحرج حالما تكون المهام قابلـة للتنفيذ، ويجب أيضاً في المقابلة الجيدة أن تسعى إلى الإقلال من التفاعلات بين الإجرائيـات وذلك بمقابلة المهام التي لها درجة عالية من التفاعل المشترك سوياً على نفس الإجرائية.

على سبيل المثال في الشكل**(6-3)** عرض لمقابلة فعالة لمخطط التقسيم و التفاعل بـين المهام الواردة في الشكل **(5-3)** إلى أربع إجرائيات، ويلاحظ في هذه الحالة أن الحد الأعلـى لأربع إجرائيات يمكن أن يستخدم بشكل مفيد على الرغم من أن عدد المهام هو سبع مهام، وهذا يعود إلى أن درجة التزامن القصوى هي أربعة فقط. أما المهام الثلاث الأخيرة فيمكن أن يتم مقابلتها.

ولكن من الأفضل مقابلة المهام التي ترتبط بضلع على نفس الإجرائية لأن ذلك يمنـع أن يحدث تفاعل بين المهام بسبب حدوث تفاعل بين الإجرائيات. فمثلاً: في الشكل (3-6.b) إذا قمنا بمقابلة المهمة 5 مع P_2 فإن ذلك يتطلب من الإجرائيتان P_0 و P_1 أن تتفاعلا مع الإجرائية P_2. وفي المقابلة الحالية فإنه يوجد تفاعل وحيد بين الإجرائيتان P_0 و P_1.



الشكل(3-6) المقابلة لمخطط المهام في شكل(3-5) إلى أربعة إجرائيات P.

3.3 تقنيات التقسيم

كما ذُكر قبل ذلك فإن إحدى الخطوات الأساسية التي نحتاجها من أجل حل المسائل بالتوازي هي **تقسيم** (Decomposition) العمليات الحسابية لتأديتها على مجموعة مهام بشكل متزامن كما هو محدد بواسطة مخطط التبعية. وسنقوم في هذه الفقرة بدراسة بعض طرق التقسيمات الشائعة من أجل الحصول على التزامن. مع العلم بأن هذه الأساليب ليست شاملة لكل تقنيات التقسيم، وكذلك فإن أي أسلوب تقسيم مذكور ليس هناك ضمان دائم بأنه سيعطي أفضل خوارزمية متوازية. وعلى الرغم من وجود بعض جوانب القصور ، فإن تقنيات التقسيم المذكورة في هذا الفصل غالبا ما تكون نقطة بداية جيدة للعديد من المسائل، وللحصول على تقسيمات فعالة لعدد كبير من المسائل فإنه يمكن استخدام أسلوب تقسيم واحد أو المزج بين أكثر من أسلوب.

تم تصنيف تقنيات التقسيم هذه إلى:

- التقسيم العَوْدِي (Recursive Decomposition).
- تقسيم البيانات (Data Decomposition).
- التقسيم الاستكشافي (Exploratory Decomposition).
- التقسيم التخميني (Speculative Decomposition).

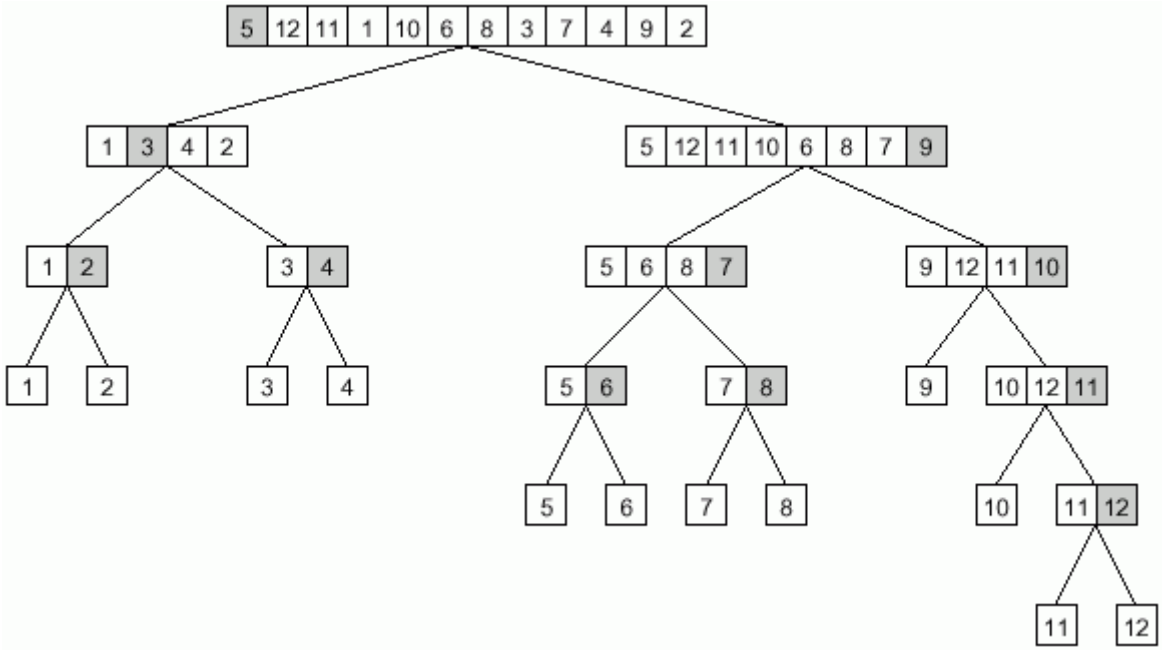
إن أسلوبي التقسيم العَودِي وتقسيم البيانات كلاهما ذو غرض عام؛ وذلك لأنه يمكن استخدامهما في حل العديد من المسائل. أما أسلوبي التقسيم التخميني و الاستكشافي فكلاهما يستخدم لأغراض أكثر خصوصية، ويمكن تطبيقهما على أنواع معينة من المسائل.

3.3.1 التقسيم العَودِي (Recursive Decomposition)

التقسيم العَودِي هو وسيلة لتحقيق التزامن في المسائل التي يمكن حلها باستخدام استراتيجية "فرّق-تَسُد"(divide-and-conquer)، وفى هذا الأسلوب فإن المسألة تُحل بتقسيمها أولا إلى مجموعة مسائل فرعية مستقلة، وكل واحدة من هذه المسائل الفرعية تُحل أيضاً بتكرار تقسيمها إلى مسائل فرعية ثم تُتبع بنتائجها مجتمعة. واستراتيجية "فَرق-تَسُد" تؤدي إلى تزامن طبيعي وذلك لأنه يمكن حل المسائل الفرعية المختلفة بنفس الوقت.

مثال(3-3) : الفرز السريع (Quicksort)

بفرض أننا نريد فرز(ترتيب) السلسلة A والمكونة من n عنصر باستخدام خوارزمية الفرز السريع شائعة الاستخدام والتي تعد خوارزميةً من النمط (فَرق تَسُد). تبدأ هذه الخوارزمية باختيار عنصر محوري X، ثم بعد ذلك يتم تقسيم السلسلة A إلى سلسلتين فرعيتين A0 و A1 بحيث تكون جميع عناصر A0 أصغر من X وكل عناصر A1 أكبر من أو تساوى X. تشكل خطوة التجزيء هذه خطوة التقسيم في الخوارزمية. وكل من السلسلتين الفرعيتين A0 وA1 يتم فرزهما بواسطة الاستدعاء العَودِي لخوارزمية الفرز السريع Quicksort. وكل استدعاء من هذه الاستدعاءات العَودية يؤدي إلى تقسيم إضافي للسلاسل.
يوضح الشكل (3-7) هذه المسألة مع فرز 12 عدد. ويلاحظ أن الاستدعاء العودي لا يتوقف إلا عندما تحتوي كل سلسلة فرعية على عنصر وحيد فقط.



الشكل(7-3): مخطط التبعية للفرز السريع والقائم على التقسيم العودي لتقسيم متسلسلة من ١٢ رقم.

في الشكل (7-3) قد عرفنا المهمة بأنا القيام بتقسيم سلسلة فرعية معطاة. وعلى هـذا فان الشكل(7-3) يوضح أيضا مخطط المهمة الخاص بالمسألة. ففي البدايـة هنـاك سلـسلة واحدة(جذر الشجرة). ويمكننا أن نستخدم عملية واحدة لتقسيمها، وعند اكتمال مهمـة الجذر فإنه ينتج عنها اثنتين من السلاسل الفرعية (A0 وA1 متوافقتين مـع العقـدتين في المستوى الأول من الشجرة) وكل منهما يمكن أن يُقسم بالتوازي، وبنفس الطريقة يـستمر التزامن في الزيادة كلما نزلنا إلى أسفل الشجرة.

في بعض الأحيان يمكن القيام بإعادة هيكلة العملية الحسابية وذلك لجعلها قابلة للتقسيم العودي حتى لو كانت الخوارزمية المستخدمة للمسألة ليست من النمط (فَرق-تَسُد). فعلـى سبيل المثال بفرض أننا نريد إيجاد العنصر الأصغر في سلسلة غير مرتبـة A مكونـة مـن n عنصر. تقوم الخوارزمية التسلسلية لحل هذه المسألة بالتدقيق في كل السلسلة A، وفي كـل

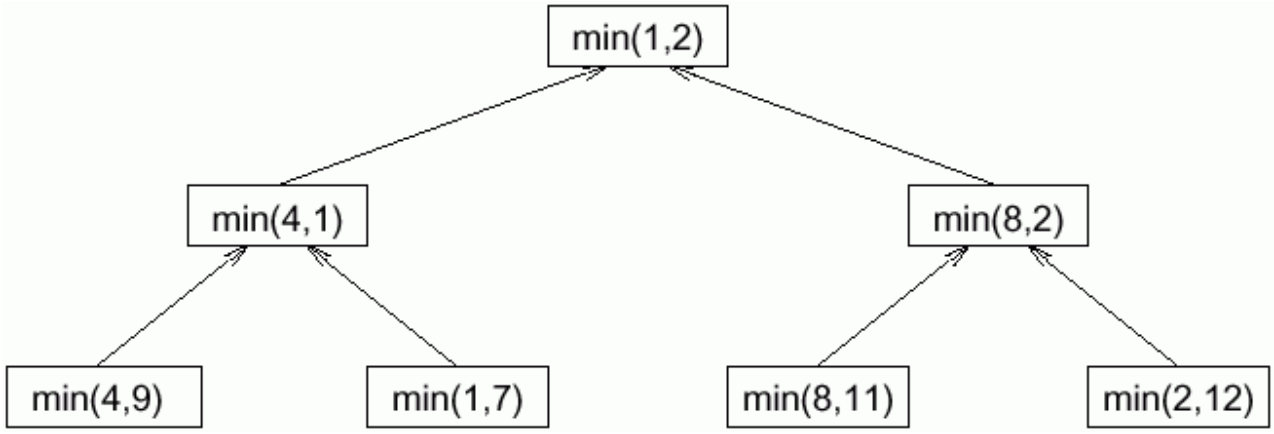
خطوة تقوم بتسجيل أصغر عنصر موجود حتى الآن، كما هو موضح في الخوارزميـة 1-3. ومن السهل ادارك أن هذه الخوارزمية ليست تزامنية.

الخوارزمية1-3: برنامج تسلسلي لإيجاد العدد الأصغر في مصفوفة أعداد A بطول n .

```
1.    procedure SERIAL_MIN (A, n)
2.    begin
3.    min = A[0];
4.    for i := 1 to n - 1 do
5.        if (A[i] < min) min := A[i];
6.    endfor;
7.    return min;
8.    end SERIAL_MIN
```

حين نُعيد هيكلة هذه الخوارزمية لكي نجعلها من النمط "فَرّق-تَسُد"، فإنه يمكـن لنـا استخدام التقسيم العودي كي نجعل منها خوارزمية متزامنة.

الخوارزمية 3-2 هي من النمط "فَرّق-تَسُد" وهي من أجل إيجاد العنـصر الأصـغر في مصفوفة، وفى هذه الخوارزمية نقوم بتقسيم السلسلة A إلى سلسلتين فرعيتين لهمـا الحجـم (n/2)، و من ثم نقوم بإيجاد العنصر الأصغر لكل واحدة من السلسلتين وذلـك باسـتخدام الاستدعاء العودي. والعنصر الأصغر الكلى يوجد بانتقاء أصغر عنصر في هذين السلـسلتين. يتوقف الاستدعاء العودي فقط عندما يتبقى عنصر واحد في كل سلسلة. و الآن و بعـد أن أعدنا هيكلة الخوارزمية التسلسلية بذا الأسلوب فإنه يكون من السهل رسم المخطط المُعتمِد على المهمة لهذه المسألة.



الشكل**(8-3)**: **مخطط التبعية لإيجاد العدد الأصغر للسلسلة [4,9,1,7,8,11,2,12]. كل عقدة في الشجرة تمثل مهمة لإيجاد العدد الأصغر من عددين.**

الخوارزمية 2-3 :برنامج عودي لإيجاد العدد الأصغر من بين عناصر A المكونة من n عدداً .

```
1.   procedure RECURSIVE_MIN (A, n)
2.   begin
3.   if (n = 1) then
4.       min := A[0];
5.   else
6.       lmin := RECURSIVE_MIN (A, n/2);
7.       rmin := RECURSIVE_MIN (&(A[n/2]), n - n/2);
8.       if (lmin < rmin) then
9.           min := lmin;
10.      else
11.          min := rmin;
12.      endelse;
13.  endelse;
14.  return min;
15.  end RECURSIVE_MIN
```

3.3.2 تقسيم البيانات (Data Decomposition)

يعتبر تقسيم البيانات أسلوباً فعّالاً وشـائعاً يـستخدم للحـصول علـى التـزامن في الخوارزميات التي تعمل على تراكيب البيانات الضخمة. وفي هـذا الأسـلوب، فالتقـسيم

للعمليات الحسابية يتم في خطوتين: **الخطوة الأولى** يتم فيها تجزيء البيانات التي تجري عليها العمليات الحسابية، وفي **الخطوة الثانية** يتم استخدام البيانات الجزئة لإحداث تقسيم للعملية الحسابية إلى مهام. والعمليات التي تتم بها هذه المهام على أجزاء البيانات المختلفة عادة ما تكون متشابهة، أو يتم اختيارها من بين مجموعة صغيرة من العمليات.

ويمكن أن يتم تقسيم البيانات بعدة طرق مختلفة كما سيأتي تفصيله بعد قليل. وبشكل عام ينبغي الاستطلاع وتقييم كل الطرق الممكنة لتقسيم البيانات ومن ثم تحديد أيها تعطي تقسيم حسابي طبيعي ومناسب.

- تجزيء **البيانات المخرجة**

يمكن في العديد من الحسابات أن يتم حساب كل عنصر من المخرجات على حدة كوظيفة للمدخلات. وفي مثل هذه العمليات الحسابية فإن تجزيء البيانات المخرجة يتسبب في حدوث التقسيم آلياً للمسألة إلى مهام، بحيث يتم إسناد عمل حساب لجزء من المخرجات إلى كل مهمة. وفي المثال**(4-3)** سنعرض لعملية ضرب مصفوفة لإيضاح التقسيم المعتمد على تجزيء البيانات المخرجة.

المثال(4-3) ضرب المصفوفات المربعة

بفرض أننا نريد إجراء عملية الضرب على المصفوفتين(A و B) وكلاهما من الحجم $n \times n$ وسنقوم بوضع الناتج في المصفوفة C. في الشكل**(9-3)** توضيح لتقسيم هذه المسألة إلى أربع مهام. حيث تم اعتبار أن كل مصفوفة مركبة من أربع كتل (أو مصفوفات جزئية) تحدد هذه الكتل بواسطة تقسيم كل بعد من المصفوفة إلى نصفين (وبذلك سينتج لدينا أربع كتل داخل المصفوفة). والمصفوفات الجزئية الأربع للمصفوفة C (كلها تقريبا من الحجم $n/2 \times n/2$) يتم حسابها مستقلة باستخدام أربع مهام كمجموع لحواصل الضرب الموافق للمصفوفات الجزئية الموجودة في A و B.

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

$$\begin{aligned} \text{Task 1: } C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ \text{Task 2: } C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ \text{Task 3: } C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ \text{Task 4: } C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

(b)

الشكل (3-9): (a) التجزيء لمصفوفات المدخلات والمخرجات إلى مصفوفات جزئية بحجم ٢×٢. (b) التقسيم لمسألة ضرب المصفوفات إلى أربع مهام إعتماداً على تجزيء المصفوفات الوارد في (a).

إن التقسيم الموضح في الشكل **(3-9)** قائم على تجزيء مصفوفة الخـرج C إلى أربـع مصفوفات جزئية وكل واحدة من المهام الأربع تقوم بحساب واحدة من المصفوفات الجزئية. ويجب ملاحظة أن تقسيم البيانات يختلف عن تقسيم العملية الحسابية إلى مهام. وبالرغم من أن كلاهما متصل بالآخر وأن الأول في الغالب مساعد للثاني، فإن تقسيماً معطى للبيانات لا ينتج عنه تقسيماً فريداً إلى مهام. على سبيل المثال الشكل **(3-10)** يوضح تقسيمين آخـرين لضرب المصفوفات، كل واحد إلى ثماني مهام، وهذان التقسيمان مـاثلان لـنفس تقـسيم البيانات الموجود في الشكل السابق**(3-9.a)**.

التقسيم I	التقسيم II
Task 1: $C_{1,1} = A_{1,1}B_{1,1}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$
Task 3: $C_{1,2} = A_{1,1}B_{1,2}$	Task 3: $C_{1,2} = A_{1,2}B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$
Task 5: $C_{2,1} = A_{2,1}B_{1,1}$	Task 5: $C_{2,1} = A_{2,2}B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$
Task 7: $C_{2,2} = A_{2,1}B_{1,2}$	Task 7: $C_{2,2} = A_{2,1}B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

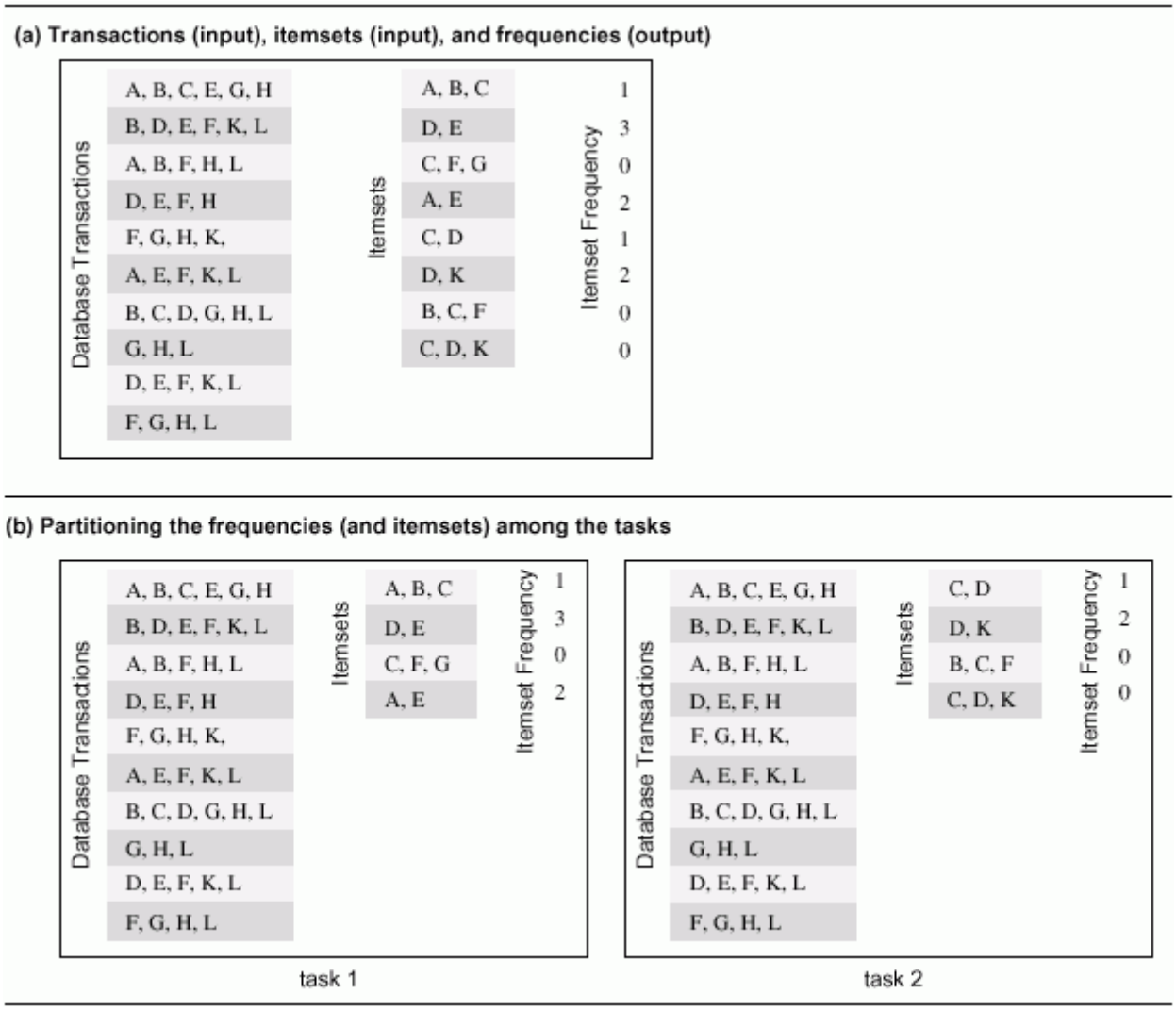
الشكل (3-10): مثالان لتقسيم عملية ضرب المصفوفة إلى ثمانية مهام.

سنعطي مثالاً آخر لتوضيح فكرة التقسيمات المعتمدة على تجزيء البيانات، وسـنعرض فيه مسألة حساب التكرار ضمن مجموعة من عدة عناصر تحدث سوياً(itemset) في قاعـدة بيانات إجرائية، والتي يمكن تقسيمها بالاعتماد على التجزيء للبيانات المخرجة.

المثال(3-5): حساب تكرارات المكونات في التعامل مع قاعدة البيانات

باعتبار أننا نريد حساب التكرار لموعة من العناصر التي تحدث سوياً(itemsets) في قاعدة بيانات تفاعلية أو إجرائية Transation Database. لدينا في هذه المسألة الموعتان T و I بحيث أن الموعة T تحتوي على n إجرائية(Transation) أما الموعة I فتحتـوي على m عدد من itemset. كل إجرائية و كل مجموعة عناصر(itemset) تحتوي على عدد قليل من العناصر من بين مجموعة ممكنة من العناصر. على سبيل المثال، قد تكون T قاعـدة بيانات متجر لمبيعات الزبائن. فإذا رغب المستودع أن يعرف عدد الزبائن الذين اشتروا كـل مجموعة العناصر المحددة، فإنه يجب أن يحسب عدد المرات التي ظهرت فيها عناصر I في جميع الإجرائيات، بمعنى أنه عدد الإجرائيات التي مجموعة العناصر جزء منها.

الشكل **(3-11.a)** يوضح مثال لهذا النوع من الحسابات. وقاعدة البيانات الموضحة في الشكل**(3-11)** تتألف من ١٠ إجرائيات، ونحن نرغب بحساب التكرار لثمانيـة مجموعـات عناصر Itemset الموضحة في العمود الثاني. التكرارات الفعلية لهذه الموعـات في قاعـدة البيانات معروضة في العمود الثالث. وللتمثيل: مجموعة العناصر {D,E} تظهـر ثـلاث مرات، مرة في الإجرائية الثانية، ومرة أخرى في الإجرائية الرابعة ومـرة ثالثـة يظهـر في الإجرائية التاسعة.



الشكل(3-11): حساب تكرار المكونات في تعاملات قاعدة البيانات

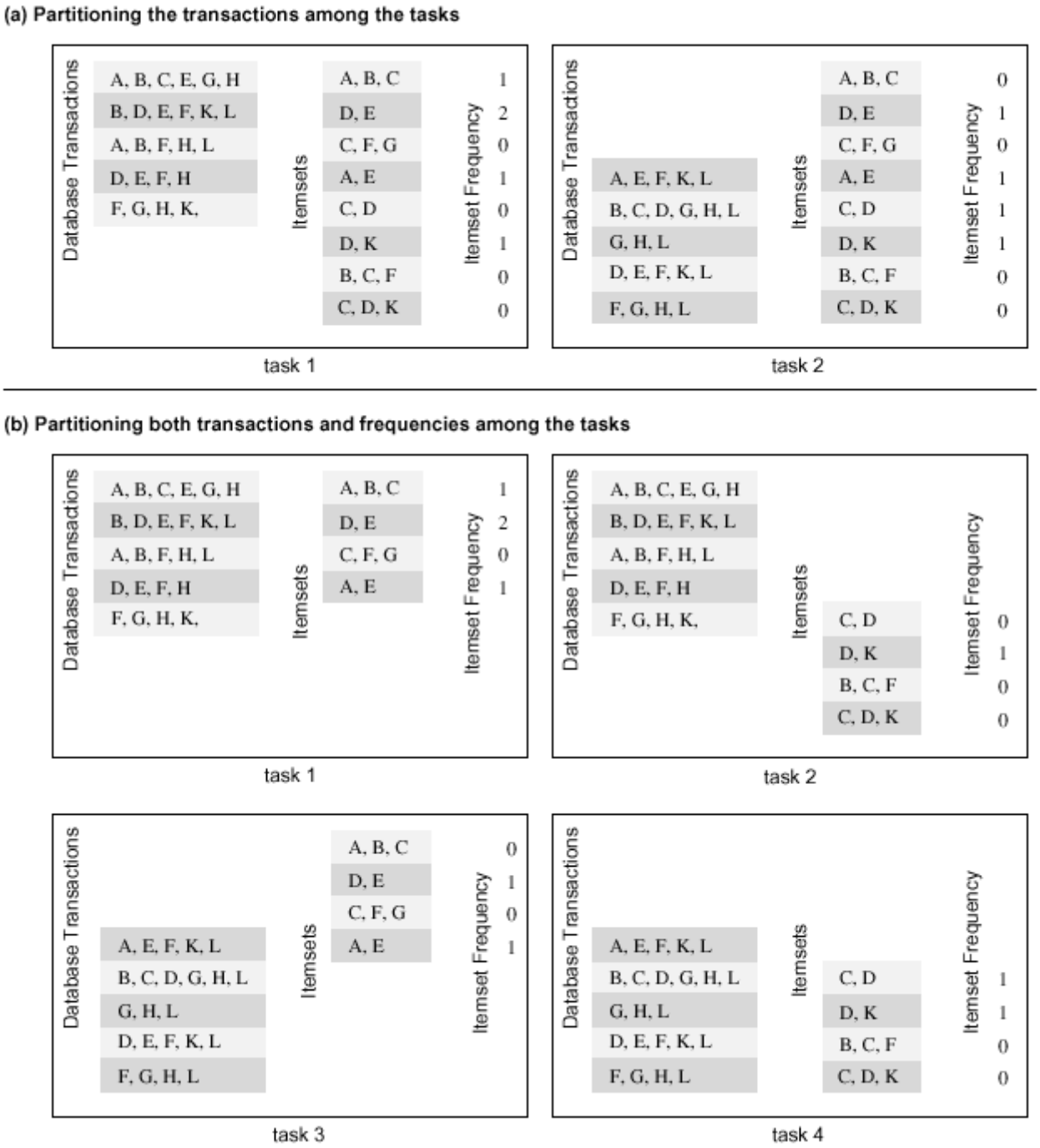
يوضح الشكل **(3-11.b)** كيف أن عملية حساب التكرار لموعة العناصر يمكن أن تقسم إلى مهمتين وذلك بتقسيم البيانات المخرجة إلى جزأين ومن ثم تحسب كل مهمة الجزء الخاص با من التكرارات. لاحظ أن مدخلات مجموعة العناصر أيضا تم تقسيمها، ولكن الداعي للتقسيم في الشكل **(3-11.b)** هو أن تقوم كل مهمة بحساب جزء من التكرارات الذي أسند إليها بشكل مستقل.

- **تجزيء البيانات المدخلة**

إن التجزيء للبيانات المخرجة يمكن أن يتم فقط إذا كان كل مخرج يمكن أن يحسب طبيعياً كوظيفة للبيانات المدخلة. وفي العديد من الخوارزميات ليس من الممكن القيام بتجزيء البيانات المخرجة. فمثلاً: عند محاولة إيجاد العدد الأصغر أو الأكبر لموعة من الأعداد، فإن المخرجات هي قيمة فردية غير معلومة. وفي خوارزمية الفرز فالعناصر الفردية من المخرجات لا يمكن معاملتها وهي منعزلة. وفي مثل هذه الحالات قد يكون من الممكن القيام بتجزيء البيانات المدخلة ومن ثم استخدام هذه الأقسام للحصول على التزامن. يتم إنشاء مهمة لكل جزء من بيانات المدخلات، وتنفذ هذه المهمة قدر المستطاع باستخدام البيانات المحلية. لاحظ أن الحل للمهام في تجزيء البيانات المدخلة ربما لا يؤدي إلى الحل النهائي مباشرة، وفي مثل هذه الحالات يلزم إجراء حسابات إضافية لتجميع النواتج الجزئية. فعلى سبيل المثال عند محاولة إيجاد مجموع سلسلة مكونة من N عدد باستخدام P إجرائية $(N>P)$، فإنه يمكن تجزيء المدخلات إلى P سلسلة فرعية بأحجام متساوية. بعد ذلك تقوم كل مهمة بحساب الموع لواحدة من السلاسل الفرعية. وفي النهاية يمكن أن نحصل على الناتج النهائي وذلك بتجميع نواتج الـ P سلسلة فرعية.

في مسألة حساب التكرار لموعة من العناصر في قاعدة بيانات إجرائية الموضحة في المثال**(3-5)** يمكن أيضاً أن يتم تقسيمها بالاعتماد على التجزيء للمدخلات. يوضح الشكل

(3-12.a) تجزيء البيانات المدخلة. تقوم كل مهمة من المهمتين بحساب التكرارات للمجموعة الفرعية الخاصة بها. الموعتان الناتجتان عن المهمتين تمثلان نواتج وسيطة وبضم هذه النواتج سوياً سينتج لدينا الناتج النهائي.



الشكل(12-3): بعض التقسيمات لحساب تكرار المكونات في تعاملات قاعدة البيانات

- **تجزيء البيانات المخرجة والمدخلة معاً**

قد يكون ممكنا في بعض الحالات التي تقبل تجزيء المخرجات أن يتم تجزيء المدخلات أيضا مما يؤدي إلى تزامن أكثر. فعلى سبيل المثال، في الـشكل(**3-12.b**) مـسألة حـساب التكرار في قاعدة بيانات. ويظهر فيـه أن Transaction مقـسم إلى جـزأين وكـذلك frequencies قد تم تقسيمها إلى جزأين، ويسند لكل مهمة أحد الاحتمالات الأربع. وبعد ذلك تقوم كل مهمة بحساب الجزء الخاص بـا من التكرارات، وفي النهاية يتم جمع مخرجـات المهمة 1 مع مخرجات المهمة 3، ويتم أيضاً جمع المهمة 2 مع المهمة 4.

3.3.3 التقسيم الاستكشافي (Exploratory Decomposition)

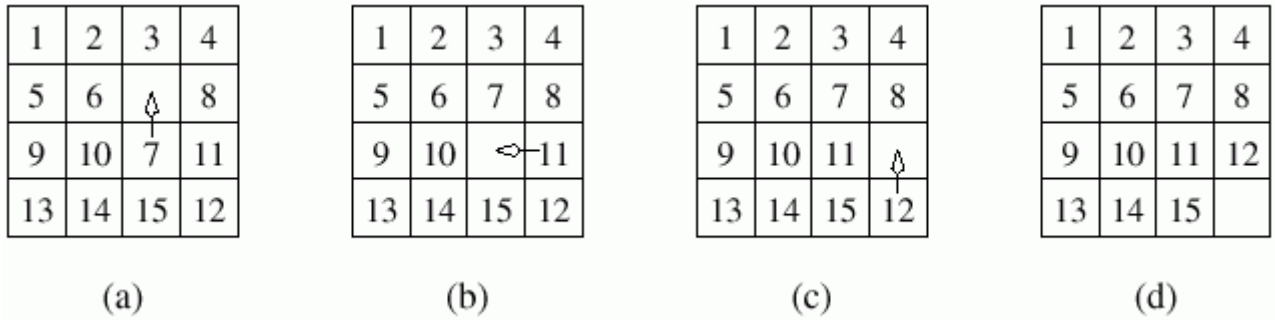
يستخدم التقسيم الاستكشافي لغرض تقسيم المسائل التي تتوافق حساباتا ضـمنيا مـع البحث عن فضاء للحلول. وفى التقسيم الاستكشافي نقوم بتقسيم مساحة البحث إلى أجزاء صغيرة، ويتم البحث في كل الأجزاء بشكل متزامن إلى أن يتم إيجاد الحل المطلوب. وتعتـبر مسألة "أحجية المربع" (15-Puzzle)كمثال للتقسيم الاستكشافي.

المثال (6-3): مسألة أحجية المربع

تتكون هذه المسألة من 15 بلاطة مرقمة من 1 إلى 15 بالإضافة إلى مساحة فارغة تكفي لبلاطة واحدة، وقد تم وضع هذه البلاط على شكل شبكة مربعـة أبعادهـا 4×4. ويمكـن تحريك بلاطة ناحية الموضع الفارغ من الموضع الـاور، وبذا سينتج فراغ في المكان الأصـلي للبلاطة التي حركت. وبناءا على ترتيب الشبكة فإنه يمكن أن يكون هناك حتى أربع حركات ممكنة للأعلى والأسفل واليمين واليسار. والترتيب الأول والأخير للبلاط يكون محدد مسبقاً.

والهدف هو تحديد أي تتابع أو أقصر تتابع للحركات يؤدي إلى التحول من الترتيب الأولي إلى الترتيب النهائي.

يوضح الشكل(13-3) نموذج للترتيب الأول والترتيب النهائي وتتابع الحركات من الترتيب الأول إلى الترتيب النهائي.



الشكل(13-3): مسألة أحجية المربع توضح الترتيب الأول (a) والترتيب النهائي (d) وتتابع الحركات من الترتيب الأول وحتى الترتيب النهائي.

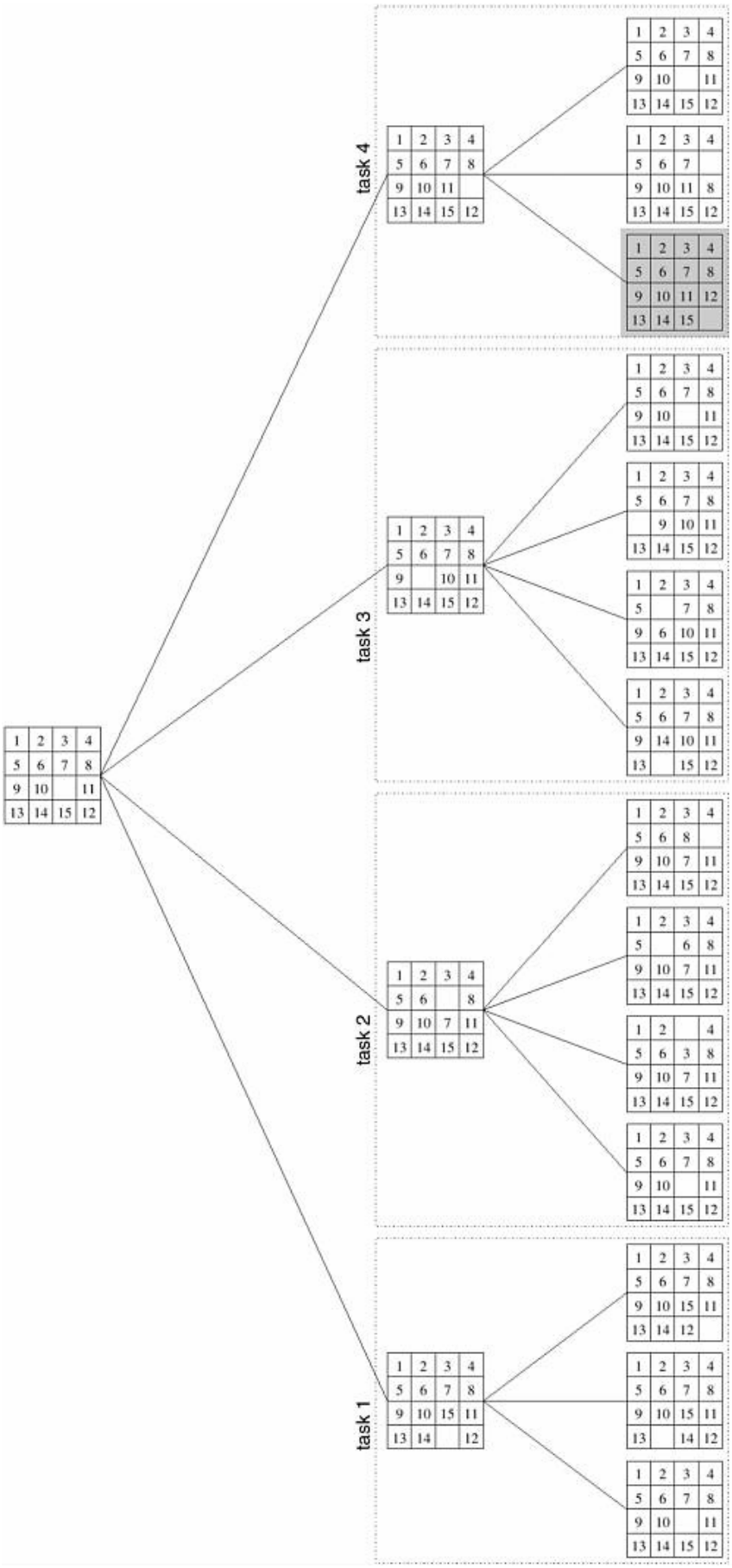
ويمكن حل مسألة أحجية المربع باستخدام تقنيات شجرة البحث. فبدءاً من الترتيب الأول تكون جميع الترتيبات المتفرعة الممكنة قد أنتجت. وربما يكون لدى الترتيب 2 أو 3 أو 4 ترتيبات وريثة أخرى، كل واحد منها يحتل الحيز الفارغ الاور له. ومهمة إيجاد مسار من الترتيب الأولي إلى الأخير تتحول الآن إلى البحث عن مسار من واحد من هذه الترتيبات الجديدة الناشئة إلى الترتيب النهائي. ونظرا لأن واحد من هذه الترتيبات الجديدة الناشئة يجب أن يكون على بعد تحريكةٍ واحدة من الحل، فإننا نكون قد حققنا تقدما ناحية إيجاد الحل. ومساحة الترتيب الناتجة عن شجرة البحث يمكن أن يشار إليها كرسم لحالة المساحة. وكل عقدة في الرسم تعتبر ترتيب وكل ضلع (الخطوط الواصلة بين الترتيبات) من الرسم يصل بين الترتيبات فيمكن الوصول بحركة واحدة للبلاطة.

إحدى الطرق لحل هذه المسألة بالتوازي هي كما يلي:

أولاً: عمل مستويات قليلة من الترتيبات بشكل تسلسلي بدءاً من الترتيب الأول حتى يكون لدى شجرة البحث عدد كاف من العقد.

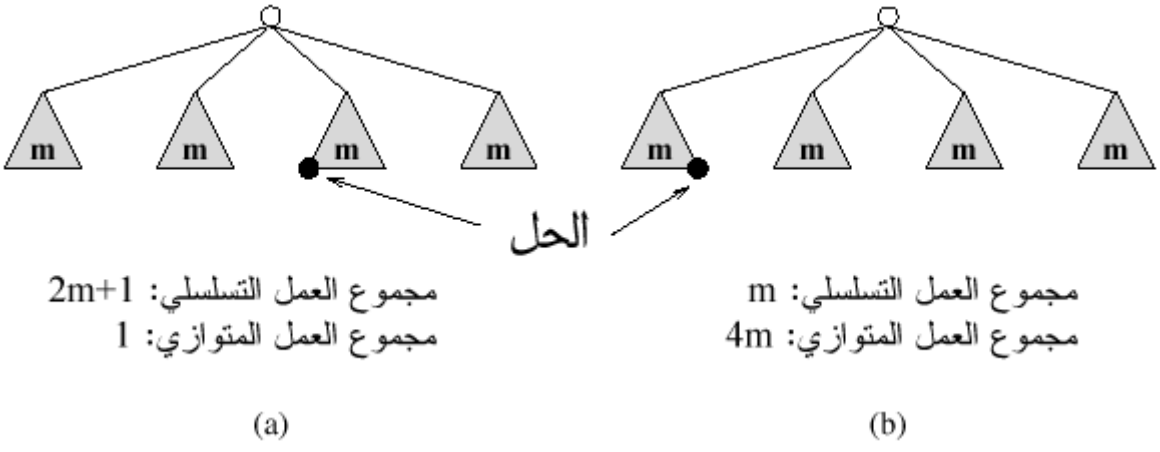
ثانياً: نبدأ بإسناد كل عقدة لمهمة وذلك لعمل استطلاع آخر حتى تصل أحد المهام إلى الحل. وبمجرد وصول أحد المهام المتزامنة إلى الحل فإنه يمكن أن تخبر البقية لإيقاف البحث.

الشكل(**14-3**) يوضح تقسيم إلى أربعة مهام والذي تتوصل فيه المهمة ٤ إلى الحل.



الشكل(14-3): الأوضاع التي تنتج عن مثال لمسألة أحجية المربع

لاحظ أن التقسيم الاستكشافي قد يبدو وكأنه مشابه لتقسيم البيانات (يمكن النظر إلى مساحة البحث على أنها البيانات التي تم تقسيمها) ولكن التقسيمين مختلفين في الجانب التالي؛ في تقسيم البيانات يتم تنفيذ المهام بتمامها فكل مهمة تؤدي حسابات مفيدة بـدف حـل المسألة. أما في التقسيم الاستكشافي فيمكن إيقاف تنفيذ المهمة ولو لم تكتمل وذلك حالمـا تتوصل أحد المهام إلى الحل النهائي. ولذا فان أقسام مساحة البحث المـستخدمة للتركيـب المتوازي يمكن أن تكون مختلفة عن التي استخدمت الخوارزمية التسلسلية. ومن الجدير ذكره أن العمل المنفذ باستخدام الصيغة المتوازية قد يكون أقل أو أكثر منه في الخوارزمية التسلسلية، فعلى سبيل المثال يمكن ملاحظة مساحة البحث التي تم تقسيمها إلى أربعة مهام متزامنة كمـا هو موضح بالشكل(3-15) . فإذا كان الحل موجودا في بداية مساحة البحث متوافـق مـع المهمة 3 (الشكل (3-15.a)) فإنه سيتم العثور عليه مباشرة في الصيغة المتوازيـة. أمـا في الخوارزمية التسلسلية فسيتم العثور على الحل بعد تنفيذ عمل متكافئ مع البحث في المساحة الكلية للمهام 1و 2 . ومن ناحية أخرى إذا كان الحل موجودا بالقرب من نايـة مـساحة البحث المتوافق مع المهمة 1 (الشكل(3-15.b)) فسوف يقوم التركيب المتوازي بعمل أربعة أضعاف العمل الذي تقوم به الخوارزمية التسلسلية ولن يؤدي إلى زيادة السرعة.



الشكل(3-15): توضيح للسرعة الغير منتظمة الناتجة عن التقسيم الاستكشافي.

3.3.4 التقسيم التخميني (Speculative Decomposition)

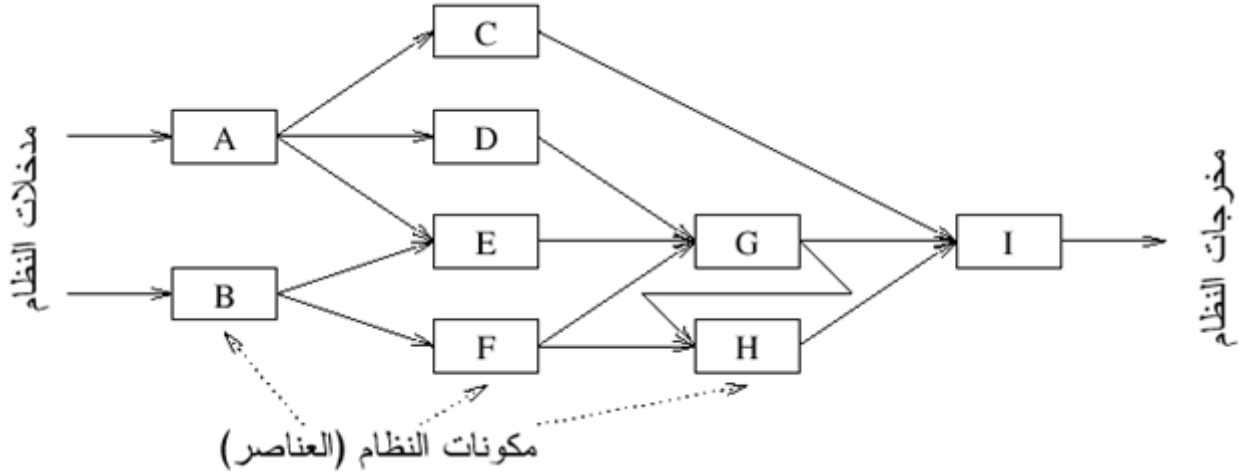
يستخدم التقسيم التخميني في البرامج التي قد تأخذ تفرع واحد من بين عدة تفرعـات حسابية اعتمادا على نتائج لعمليات حسابية سابقة. وفي هذه الحالة، فإنه أثناء تنفيـذ أحـد المهام لعملية حسابية لها ناتج يستعمل في عملية حسابية لاحقة، فيمكن لمهام أخرى أن تبدأ الحساب للمرحلة القادمة بشكل متزامن. يتشابه هذا السيناريو مع عملية تقييم (أو اختبـار) واحد أو أكثر من التفرعات الخاصة بعبارة switch في لغة C بشكل متزامن وذلك قبـل أن يكون الدخل لعبارة switch جاهزا. وفي أثناء تنفيذ أحد المهام للعملية الحسابية التي ستحل switch فمهام أخرى يمكن أن تلتقط التفرعات المتعددة لعبارة switch بـالتوازي. وعنـد الانتهاء من حساب الدخل لعبارة switch سيتم استخدام التفرع الصحيح بينما يتم إهمـال التفرعات الأخرى. إن زمن التشغيل المتوازي أقل منه في التسلسلي بمقدار الزمن اللازم لتقييم الشرط الذي تعتمد عليه المهمة القادمة بسبب أن هذا الزمن يمكـن الاسـتفادة منـه لأداء حسابات مفيدة للمرحلة التالية بالتوازي. وعلى أي حال فإن صيغة التوازي لعبارة switch تؤدي لبعض الهدر في الحساب. ومن أجل إقلال هذا الهدر في الحساب فإنه يمكن اسـتعمال صيغة معدلة قليلا من التقسيم التخميني وخصوصا في الأوضاع التي يكون فيها أحد النتـائج أكثر احتمالا من النتائج الأخرى. وفي هذه الحالة سيتم أخذ التفرع الأكثر احتمالاً بالتوازي مع العملية الحسابية السابقة. وفي حال كانت النتائج مختلفة عن المتوقع فإن العملية الحـسابية ترتد للوراء ويتم اختيار التفرع الصحيح.

والتسريع الذي يمكن أن ينتج عن التقسيم التخميني يمكن أن يكون كبيراً إذا كان هنالك مراحل تخمينية متعددة. وكمثال لتطبيق يكون فيه التقسيم التخميني نافعا : محاكاة الحـدث المتقطع، و سنعطي لمحة مبسطة لهذه المسألة.

المثال(7-3): توازي محاكاة الحدث المتقطع

باعتبار أن لدينا محاكاة لنظام ممثل بشبكة أو بيان موجه، بحيث تمثل العقـد في هـذه الشبكة عناصر ، وكل عنصر له عازل مدخلات للوظائف ، والحالة الأولية لكل عنـصر أو عقده تكون عاطلة ، كل عنصر عاطل يقوم بأخذ الوظائف من طابور المدخلات ، فإذا كان هنالك وظيفة فإنه يعالجها بكمية محددة من الوقت ، ومن ثم يضعها في عازل الدخل للعنصر المرتبط معه من طرفه الخارج، قد ينتظر العنصر بعض الوقت إذا كان عازل المدخلات لأحد جيرانه غير فارغ ويكون الانتظار حتى يقوم الجار بالتقاط الوظيفة ليترك مساحة فارغـة في العازل. هنالك عدد محدد من أنواع وظائف المدخلات. مخرجات العنصر والوقـت الـلازم لمعالجة الوظيفة هو العمل لوظيفة المدخلات .

المسألة : محاكاة عمل الشبكة لسلسلة معطاة من ا لوظائف الداخلة وحساب إجمالي الوقـت والمظاهر الأخرى المحتملة لسلوك النظام . يوضح الشكل(3-16) شبكة مبسطة لحـل مـسألة الحدث المتقطع.



الشكل(3-16):شبكة مبسطة لمحاكاة الحدث المتقطع.

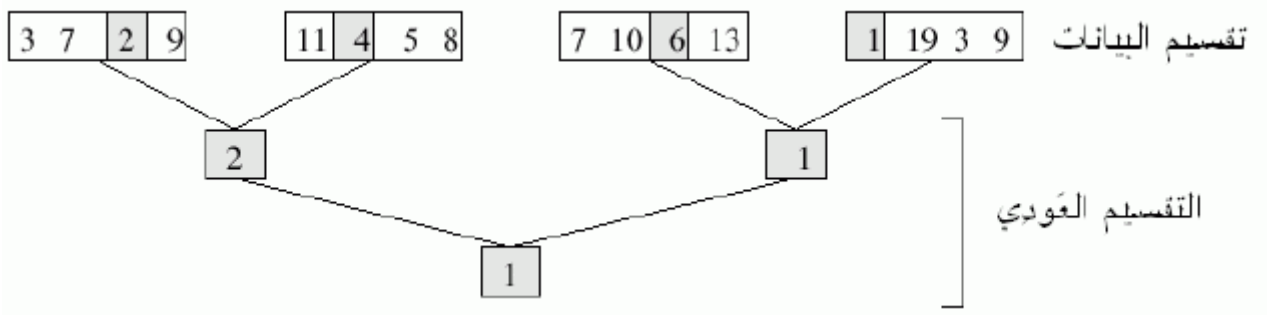
ومسألة محاكاة تتابع مدخل الوظائف على الشبكة الموصوفة في المثال (3-7) يبدوا وكأنـا أساساً متتابعة لأن المدخل لنفس المكون هو المخرج للأخر. ومع هذا فيمكننا التعريف لمهـام تخمينية تبدأ بمحاكاة الجزء الفرعي من الشبكة، وكل منها يمثل مدخلات عديدة ممكنـة في هذه المرحلة. وعندما يصبح المدخل الفعلي لمرحلة ما متوفر (نتيجة لتكملة مهمة اختيار من المرحلة السابقة) عندئذ يتم إنهاء العمل المطلوب لمحاكاة هذا المـدخل إذا كـان الـتخمين صحيحاً , أو يعاد بدء محاكاة هذه المرحلة مع المدخل الأصح إذا كان التخمين غير صحيح.

والتقسيم التخميني يختلف عن التقسيم الاستكشافي في النقاط التالية:

- في التقسيم التخميني يكون مدخل الفرع الذي يؤدى إلى عدة مهام متوازيـة غـير معروف، بينما في التقسيم الاستكشافي يكون مخرج المهام الناتج من الفرع غير معروف.
- في التقسيم التخميني تؤدي الخوارزمية التسلسلية مهمة واحدة محـددة في مرحلـة التخمين لأنا عندما تصل بداية المرحلة فإنا تعرف بالضبط ما الفرع الذي تأخذه. وبالمبادرة بحساب إمكانيات التضاعف التي يتحقق واحد منها فقط ، والبرنامج المتـوازي المـستخدم للتقسيم التخميني يؤدي عمل واحد مجمل أكثر من نظرية المسلسل.
- وحتى لو تم استكشاف أحد الإمكانيات بالتخمين فإن الخوارزمية المتوازية يمكـن أن تؤدي قدراً أكبر أو نفس القدر من العمل من الخوارزمية التسلسلية.
- ومن ناحية أخرى ففي التقسيم الاستكشافي يمكن للخوارزمية التسلسلية استكشاف بدائل مختلفة واحدة تلو الأخرى لأن الفرع الذي يمكن أن يوصل إلى الحل غـير معـروف مسبقا. ولذلك فان البرنامج المتوازي يمكن أن يؤدى نفس العمل العمل أو أقـل أو أكثـر بالمقارنة بالخوارزمية التسلسلية المعتمدة على موقع الحل في مساحة البحث.

3.3.5 **التقسيم المختلط** (Hybrid Decompositions)

نوقش عدد من طرق التقسيم التي يمكن استخدامها للحصول على صيغ متزامنة للعديـد من الخوارزميات، وتقنيات التقسيم التي مرت معنا ليست حصرية الاستخدام أي أنه يمكن – في الغالب- أن يتم جمعها معا، وغالباً فإن العمليات الحسابية تكون مركبة من عدة مراحل، وفي بعض الأحيان يكون من الضروري تطبيق أنواع مختلفة من التقسيم في المراحل المختلفة، فعلى سبيل المثال عند البحث عن العدد الأصغر ضمن مجموعة كبيرة مكونة من **N** عدد فإن التقسيم العودي تماما قد ينتج عنه مهام أكثر من عدد الإجرائيات **P** المتوفرة(يمكـن اعتبـار الإجرائية على أنها معالج)، لذلك فإن التقسيم الكفؤ يكون بتقسيم المـدخلات إلى **P** جـزء متساوي ويجعل كل مهمة تقوم بحساب العدد الأصغر ضمن السلسلة التي خصـصت لهـا. وبعد ذلك يمكن الحصول على الناتج النهائي بإيجاد العدد الأصغر من النتائج المتوسطة لـ **P** وذلك باستخدام التقسيم العودي،كما في الشكل **(3-17)**.



الشكل(3-17) التقسيم المختلط لإيجاد العدد الأصغر لمصفوفة من الحجم ١٦ باستخدام أربعة مهام.

مثال آخر لتوضيح فكرة التقسيم المختلط. بفرض أننا نريد تنفيذ الفرز السريع بـشكل متزامن. في المثال(3-3) الذي أوردناه عند شرح التقسيم العودي استخدمنا التقسيم العـودي

لاشتقاق صيغة متزامنة من الفرز السريع. ينتج عن هذه الصيغة عدد O(n) مهام لفرز سلسلة من الحجم n. ولكن بسبب اعتماد هذه المهام على بعضها وعدم تساوي الحجم بينها فـإن التزامن الفعال يعتبر محدود جدا. فعلى سبيل المثال، أول مهمة لتقسيم قائمة المـدخلات إلى قسمين تستغرق مدة O(n)، والتي تضع الحد الأعلى لمستوى الأداء المحتمل عن طريق التوازي. ولكن خطوة تقسيم القوائم التي يتم أدائها بمهام التوازي من النوع السريع يمكن تقـسيمها باستخدام أسلوب تقسيم المدخلات الذي تمت مناقشته، والتقسيم المختلط الناتج تم فيه الجمع بين التقسيم العودي وتقسيم بيانات المدخلات وهذا يؤدى إلى صيغة متزامنة بدرجة كـبيرة لخوارزمية الفرز السريع.

3.4 أمثلة للخوارزميات المتوازية

سوف نتطرق لبعض أمثلة الخوارزميات المتوازية، وسوف تكون الطريقة العامة لعـرض الخوارزمية هي: عرض الصيغة التسلسلية للخوارزمية ثم مناقشة كيفية جعلها متوازية.

3.4.1 خوارزمية الفرز الفقاعي وتوابعها (Bubble Sort).

تقوم الخوارزمية التسلسلية للفرز الفقاعي أو كما يسمى أحياناً بالفرز بالتعويم (bubble sort) بمقارنة واستبدال العناصر المتجاورة في السلسلة التي سترتب. ليكن لدينا السلسلة $< a_1, a_2, ..., a_n >$، تقوم الخوارزمية أولاً بإجراء n-1 عملية "مقارنة-واستبدال" في الترتيب التالي: $(a_1, a_2), (a_2, a_3), ..., (a_{n-1}, a_n)$. هذه الخطوة تزيح العنصر الأكبر إلى نهاية السلسلة. بعد ذلك سيتم تجاهل العنصر الأخير لأنه أخذ الترتيب الصحيح له، ثم سيتم تكـرار إعـادة عمليـة المقارنة-والاستبدال على السلسلة الناتجة وفي كل تكرار يتم إزاحة العنصر الأكبر إلى آخـر موضع في السلسلة لم يتم تجاهله. وستكون السلسلة مرتبة بعد عدد n-1 مـن التكـرارات. يمكن لنا تحسين أداء خوارزمية الفرز الفقاعي وذلك بالإنهاء عندما لا يكون هناك اسـتبدال خلال التكرار. خوارزمية الفرز الفقاعي معروضة في خوارزمية (3-3)، مع ملاحظة أنّ العبارة compare-exchange يقصد بها مقارنة العنصرين الذين مرراً لها فإذا لم يكن ترتيبهما سـليما حينها يتم استبدالهما.

إن التكرار ضمن الحلقة الداخلية في خوارزمية الفرز الفقاعي تأخذ من الوقـت $\Theta(n)$، ويتم أداء ما مجموعه $\Theta(n)$ تكرار(بسبب الحلقة الخارجية)، وبالتالي ستكون درجة التعقيـد للفرز الفقاعي مساوية إلى $\Theta(n^2)$.

من الصعوبة بمكان جعل خوارزمية الفرز الفقاعي متوازية، ولبرهان ذلك، فكر كيـف سيتم أداء عمليات المقارنة-والاستبدال أثناء كل مرحلة من الخوارزمية (السطرين 4 و 5 من خوارزمية 3-3). تقوم خوارزمية الفرز الفقاعي بمقارنة جميع الأزواج المتجـاورة بالترتيـب؛ ولهذا السبب فهي بالدرجة الأولى خوارزمية تسلسلية. وفي القسم التالي سنعرض أحد أنواع الفرز الفقاعي والتي بالإمكان أن يتم جعلها متوازية.

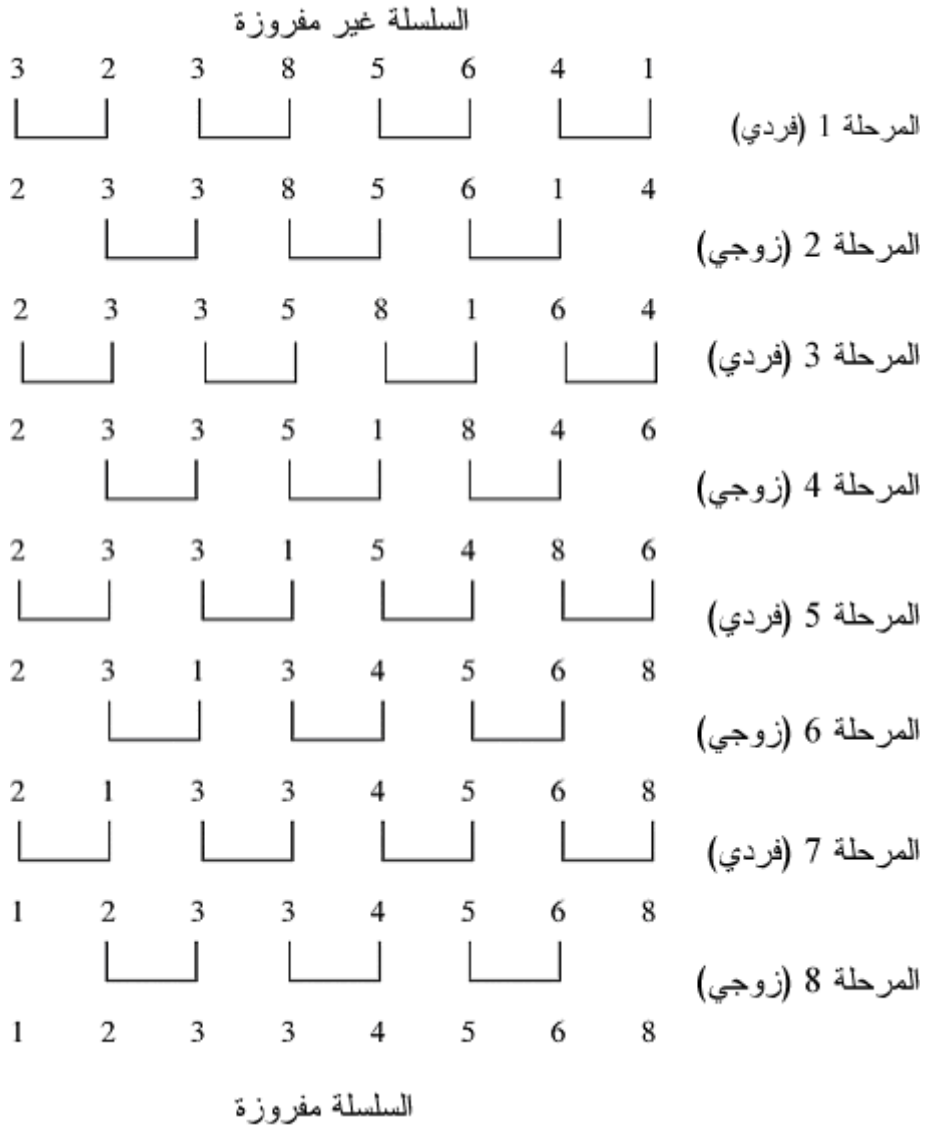
خوارزمية (3-3): خوارزمية الفرز الفقاعي التسلسلي.

```
1.   procedure BUBBLE_SORT(n)
2.   begin
3.      for i := n - 1 downto 1 do
4.          for j := 1 to i do
5.              compare-exchange(a_j, a_j + 1);
6.   end BUBBLE_SORT
```

3.4.1.1 الإبدال الزوجي-الفردي (Odd-Even Transposition)

تقوم خوارزمية "الإبدال الزوجي-الفردي" بفرز n عنصر في n مرحلة (n زوجـي)، كل مرحلة تتطلب n/2 من عمليات المقارنة-والاستبدال. وهذه الخوارزمية تتنـاوب بـين مرحلتين وهما مرحلة الفردي ومرحلة الزوجي. بافتراض أننا نريد ترتيب السلسلة $< a_1, a_2, ..., a_n >$. فخلال مرحلة الفردي، سيتم مقارنة العناصر ذوات الدليل الفردي مع ما يجاورها إلى اليمين، فإذا لم يحققا شرط الترتيب فإنه يتم إبدال أماكنهما؛ وبالتالي، فالأزواج $(a_1, a_2), (a_3, a_4), ..., (a_{n-1}, a_n)$ تقارن-وتستبدل (بفرض أن n زوجية). وعلى نحو مشابه فإنه خـلال

المرحة الزوجية، سيتم مقارنة العناصر التي لها دليل زوجي مع ما يجاورها ناحية اليمين، فإذا لم يحققا شرط الترتيب فإنه يتم إبدال أماكنهما؛ وبالتالي، فالأزواج ($(a_2, a_3), (a_4, a_5), ..., (a_{n-2}, a_{n-1})$ يتم مقارنتها-واستبدالها. وبعد n مرحلة فإن السلسلة تكون قد رتبت بالفعـل. كـل مرحلة من الخوارزمية (فردية أو زوجية) تتطلب $\Theta(n)$ عملية مقارنة، وبما أن لدينا عدد n من المراحل فلذلك ستكون درجة التعقيد للخوارزميـة هـي $\Theta(n^2)$. يوضـح الـشكل(3-18) خوارزمية الإبدال الزوجي-الفردي من خلال مثال.



الشكل(3-18)**: فرز 8 عناصر (n=8) باستخدام خوارزمية الإبدال الزوجي-الفردي، خلال كل مرحلـة هنـاك 8 عناصر يتم مقارنتها(n=8)**

الخوارزمية (3-4): الخوارزمية التسلسلية للإبدال الفردي-الزوجي.

```
1.   procedure ODD-EVEN(n)
2.   begin
3.      for i := 1 to n do
4.      begin
5.         if i is odd then
6.              for j := 0 to n/2 - 1 do
7.                  compare-exchange(a2j + 1, a2j + 2);
8.         if i is even then
9.              for j := 1 to n/2 - 1 do
10.                 compare-exchange(a2j, a2j + 1);
11.     end for
12.  end ODD-EVEN
```

الصيغة المتوازية لخوارزمية الإبدال الزوجي-الفردي:

إنه من السهل أن نقوم بجعل خوارزمية الفرز بالإبدال الزوجي-الفردي متوازية، فخلال كل مرحلة من الخوارزمية يتم إجراء عملية مقارنة-واستبدال بين عدة أزواج من العناصر بنفس الوقت. بفرض أن لدينا الحالة "عنصر واحد لكل إجرائية". وبفرض أن n هو عدد الإجرائيات (أيضا n هو عدد الأعداد التي نريد فرزها). وبافتراض أن الإجرائيات مرتبة في مصفوفة أحادية البعد. ففي البداية سيستقر العنصر a_i في العملية p_i حيث i= 1,2,3,...,n . خلال المرحلة الفردية ستقوم كل إجرائية لها دليل فردي بإجراء عملية مقارنة-واستبدال لعناصرها مع العناصر المستقرة في جارتا اليمنى. وعلى نحو مشابه، فخلال المرحلة الزوجية ستقوم كل إجرائية دليلها زوجي بإجراء عملية مقارنة-واستبدال لعناصرها مع العناصر المستقرة في جارتا اليمنى. هذه الصيغة المتوازية معروضة في الخوارزمية (3-5).

الخوارزمية(5-3):الصيغة المتوازية لخوارزمية الفرز بالإبدال الزوجي-الفردي، على عدد n-عملية بشكل حلقة.

```
1.   procedure ODD-EVEN_PAR (n)
2.   begin
3.      id := process's label
4.      for i := 1 to n do
5.      begin
6.         if i is odd then
7.             if id is odd then
8.                 compare-exchange_min(id + 1);
9.             else
10.                compare-exchange_max(id - 1);
11.        if i is even then
12.            if id is even then
13.                compare-exchange_min(id + 1);
14.            else
15.                compare-exchange_max(id - 1);
16.     end for
17.  end ODD-EVEN_PAR
```

خلال كل مرحلة من الخوارزمية، فالعمليات الزوجية أو الفردية تؤدي خطوة مقارنة-واستبدال مع الجار الأيمن. وذلك يتطلب من الوقت $\Theta(1)$، وإجمالاً سيتم أداء n مرحلة مماثلة. فلذا، سيكون وقت التشغيل للصيغة المتوازية هو $\Theta(n)$. وبسبب أن درجة التعقيـد لأفـضل خوارزمية فرز تسلسلية لفرز n عنصر هو $\Theta(n \log n)$، فإن هذه الصيغة من الفرز بالإبـدال الزوجي-الفردي ليست مثالية الكلفة، بسبب أن ناتج عملية التشغيل هو $\Theta(n^2)$.

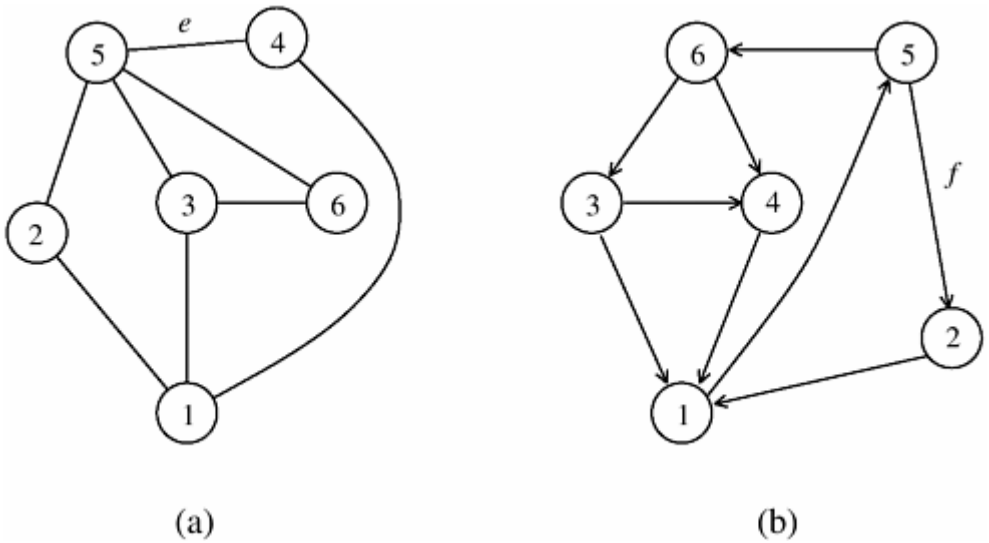
للحصول على كلفة مثالية للصيغة المتوازية، نستخدم إجرائيات أقل. لنفرض أن p هـو عدد الإجرائيات، حيث $p<n$. وفي البداية يخصص لكل إجرائية كتلة مكونة من n/p عنصر، والتي يتم ترتيبها داخلياً (باستخدام الفرز السريع أو الفرز بالدمج).

3.4.2 خوارزمية بريم Prim لإيجاد أصغر شجرة هيكلية

تلعب نظرية البيان (Graph Theory) دوراً هاماً في علم الحاسب الآلي لأنا توفر طريقة سهلة ومنهجية لنمذجة العديد من المسائل. ويمكن التعبير عن العديد من المسائل من خلال البيان (Graph) كما يمكن حلها باستخدام خوارزميات بيانية قياسية.

3.4.2.1 تعاريف ومفاهيم أساسية

إن البيان(Graph) هو الثنائية G=(V,E)،ويتألف البيان G من مجموعة من الرؤوس V، ومجموعة من الأضلاع E، بحيث أن كل ضلع يصل بين رأسين من الرؤوس. يوجد نوعان من البيان: بيان موجه (directed graph)، وبيان غير موجه (undirected graph)، في البيان الموجه يكون لكل ضلع اتجاه واحد فقط، في حين أن البيان الغير موجه يكون للضلع اتجاهين، فمثلاً وعلى افتراض أن الرأسين u و v ينتميان إلى مجموعة الرؤوس V، وكان هنالك ضلع e يصل بين الرأسين (u,v) ففي البيان الغير موجه نقول أن الرأسين u و v متصلان. أما في البيان الموجهة فنقول أن هناك اتصال من u إلى v.



الشكل (3-19) **(a) بيان غير موجه، (b) بيان موجه.**

إذا كان (u, v) ضلع في بيان غير موجه $G = (V, E)$ فإنه يقال عن الرأسين u و v أنهما مجاوران لبعضهما البعض. أما في حال كان الضلع في بيان موجه فإننا نقول أن الرأس v مجاور للرأس u(وليس العكس صحيحا).

إن **المسار** من الرأس v إلى الرأس u هو تتابع $<v_0, v_1, v_2, ..., v_k>$ من الرؤوس بحيث أن $v_0 = v$ و $v_k = u$، وأن (v_i, v_{i+1}) تنتمي إلى E من أجل $i = 0, 1, ..., k - 1$. ويُعرفُ طول المسار بأنه عدد الأضلاع الموجودة في المسار(أي التي تكون المسار).

يقرن في بعض الأحيان وزن لكل ضلع من E. والوزن في الغالب عدد حقيقي يمثل كلفة أو منفعة العبور للضلع. والبيان الذي له أوزان ترتبط مع كل ضلع يدعى بأنه بيان موزون، ويمكن أن يشار إليه $G = (V, E, w)$، حيث V هي الرؤوس و E هي الأضلاع كما أشرنا قبل قليل، أما $w:E \rightarrow R$ فهي تابع حقيقي معرف على E. ويمكن تعريف وزن البيان على أنه مجموع أوزان أضلاعه. أما وزن المسار فهو مجموع أوزان الأضلاع المكونة له.

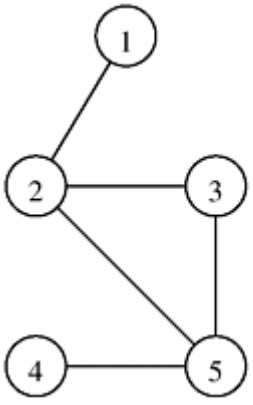
هناك طريقتان قياسيتان لتمثيل المخططات البيانية في برامج الحاسب. الأولى باستخدام المصفوفات Matrix، والثانية باستخدام القوائم المتصلة Linked List. ولأننا لن نستخدم طريقة القوائم المتصلة فلن نتطرق إليها.

ليكن لدينا البيان $G = (V, E)$ وفيه n رأس مرقمة من 1 وحتى n. إن مصفوفة الجوار adjacency matrix لهذا البيان هي المصفوفة $A=(a_{i,j})$ ولها الحجم $n \times n$، ومعرفة كالتالي:

$$a_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

في الشكل(3-20) توضيح لبيان غير موجه ممثل بمصفوفة جوار. ويلاحظ أن مصفوفة الجوار للبيان الغير موجه هي مصفوفة متناظرة. ويمكن أن يعدّل التمثيل في مصفوفة الجوار لتتماشى مع البيانات الموزونة(weighted graphs). وفي هذه الحالة يمكن تعريف $A = (a_{i,j})$ على النحو الآتي:

$$a_{i,j} = \begin{cases} w(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

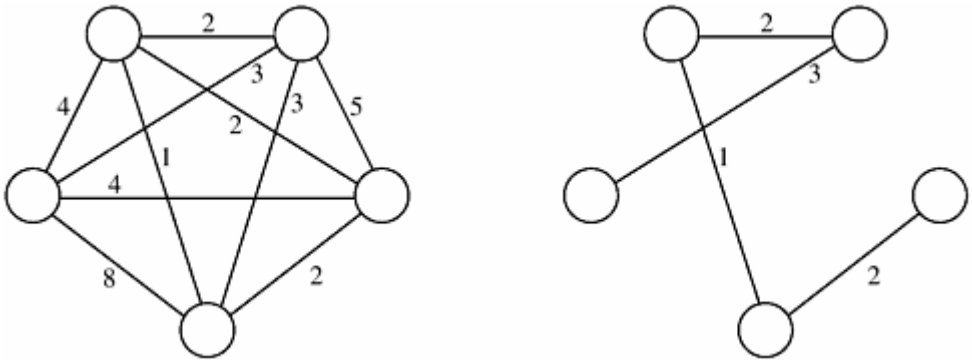
الشكل(3-20): بيان غير موجه وتمثيله بمصفوفة الجوار.

سنشير لمصفوفة الجوار المعدلة بمصفوفة جوار موزونة. والمساحة المطلوبة لتخزين مـصفوفة الجوار لبيان بعدد n رأس هي $\Theta(n^2)$.

3.4.2.2 الشجرة الهيكلية الأصغر: (خوارزمية بريم)

Minimum Spanning Tree (MST): Prim's Algorithm

إن الشجرة الهيكلية لبيان غير موجه G هي بيان جزئي من G ُيكون شجرة تحتوي على جميع رؤوس G. وفي البيان الموزون، يكون الوزن للبيان الجزئي هو مجموع أوزان الأض ـلاع فيه، والشجرة الهيكلية الأصغر (MST) لبيان موزون غير موجه هي شجرة هيكلية لها الوزن الأصغر. وتتطلب العديد من المسائل إيجاد الشجرة الهيكلية الأصغر لبيان غير موجه، فعل ـى سبيل المثال، قد يكون من الضروري إيجاد الطول الأقصر للكيابل التي تربط مجموعة حاسبات في شبكة، أو مثلاً تحديد أقل كلفة لربط خطوط الاتصالات بين المدن، وإيجاد ذلك يمكن أن يتم عن طريق البحث عن الشجرة الهيكلية الأصغر للبيان الغير موجه الذي يحتوي على ك ـل الارتباطات. في الشكل(21-3) عرض لأصغر شجرة هيكلية لبيان غير موجه.



الشكل(21-3): بيان غير موجه، والشجرة الهيكلية الأصغرية فيه.

إذا لم يكن G متصلاً، فإنه لا يمكن أن يكون له شجرة هيكلية. بل سيكون لديه غابة هيكلية spanning forest، ومن اجل تبسيط فكرة حساب أقصر شجرة هيكلية نفرض أن G متصل.

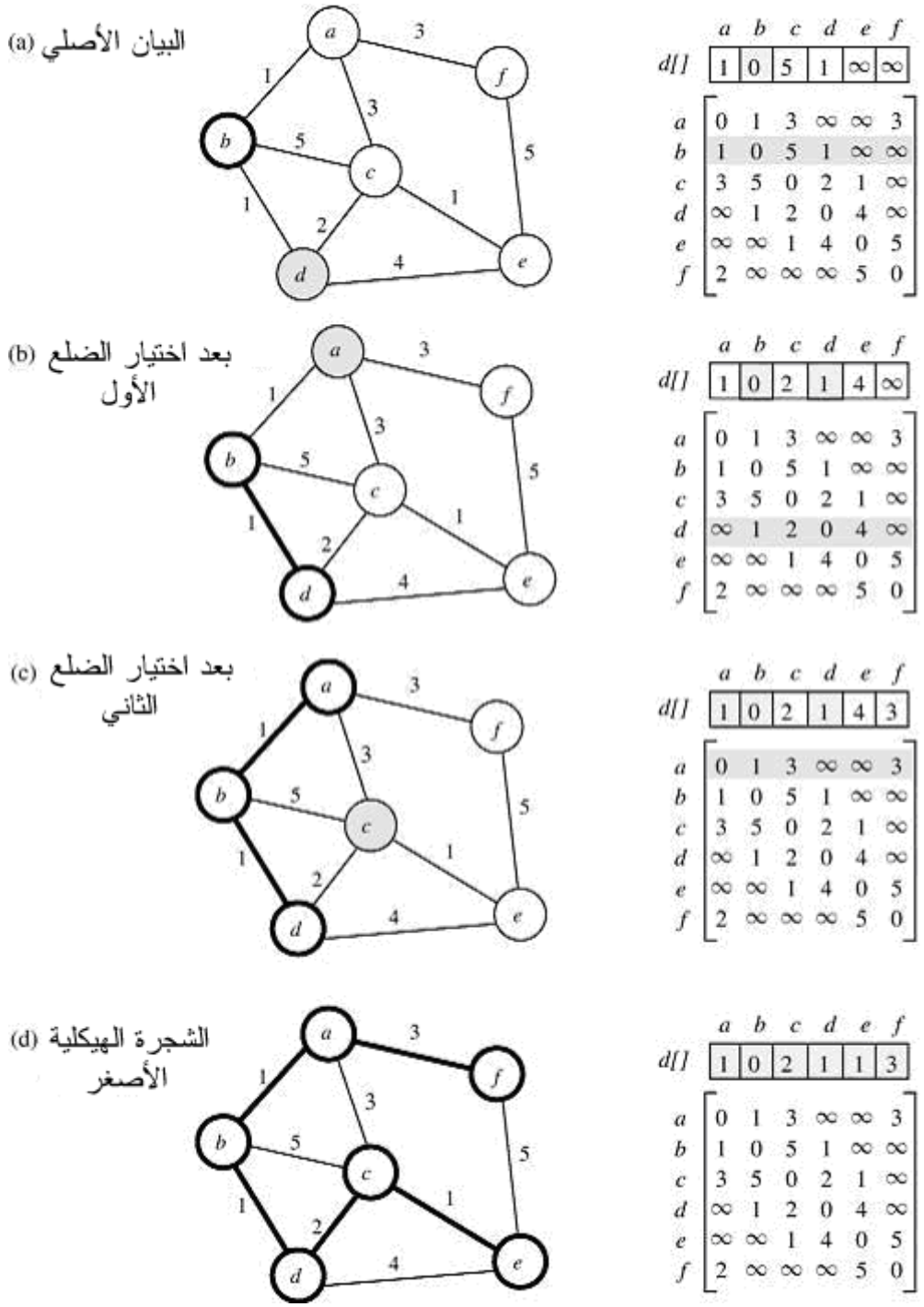
تعتبر خوارزمية بريم لإيجاد الشجرة الهيكلية الأصغر من الخوارزميات الجشعة (greedy algorithm). فهي تبدأ باختيار أحد الرؤوس عشوائيا كعنصر بداية، ثم بعد ذلك تقوم الخوارزمية بتنمية أصغر شجرة هيكلية عن طريق اختيار رأس وضلع جديدين على أن يكون في اختيارهما ضمان كونما يحققان شجرة هيكلية بأقل كلفة. ثم تستمر الخوارزمية في العمل إلى أن تكون جميع الرؤوس قد تم اختيارها.

بفرض أن G=(V,E,w) بيان موزون غير موجه نريد إيجاد الشجرة الهيكلية الأصغر فيه، وبفرض أن $A=(a_{i,j})$ هي مصفوفة الجوار الموزونة للبيان G. في الخوارزمية(3-6) عرض لخوارزمية بريم. تستخدم هذه الخوارزمية المجموعة V_T للاحتفاظ برؤوس أصغر شجرة هيكلية أثناء تكوينها. أيضا تستخدِم الخوارزمية المصفوفة d[1..n] للاحتفاظ بأوزان أضلاع الشجرة، فلكل رأس يحقق الشرط بأن v ينتمي إلى ($V- V_T$) فإن d [v] تحتفظ بوزن الضلع الذي له الوزن الأقل من أي ضلع يربط بين أي رأس في V_T إلى الرأس v.

مبدئيا تحتوي المجموعة V_T على رأس r عشوائي ليكون هو الجذر لأصغر شجرة هيكلية. أيضا فإن d[r] = 0، ولكل v من $v \in (V - V_T)$, d[v] = w(r, v) إذا كان يوجد ضلع؛ وإلا ستكون d[v] =∞.

خلال كل تكرار في الخوارزمية، يتم إضافة رأس جديد u إلى المجموعة V_T بحيث $d[u] = \min\{d [v] \mid v \in (V - V_T)\}$. وبعد أن يتم إضافة هذا الرأس فجميع قيم d[v] التي تحقق $v \in (V - V_T)$ يتم تحديثها لأنه ربما يكون هنالك الآن ضلع له وزن أقل يصل بين الرأس v والرأس المضاف u. تتوقف الخوارزمية عن العمل عندما يكون V_T =V.

يوضح الشكل(3-22) هذه الخوارزمية، حينما تتوقف خوارزمية بريم، فإن الكلفة لأصغر شجرة هيكلية هي $\sum_{v \in V} d[v]$ ، وبسهولة يمكن التعديل على الخوارزمية(3-6) لكي تخزن الأضلاع التي تتوضع على الشجرة الهيكلية الأصغرية.



الشكل(22-3)**: خوارزمية بريم لأصغر شجرة هيكلية. الجذر لأصغر شجرة هيكلية هو b. ومن أجل كل تكرار فيوضح بالخط الغامق الرؤوس التي في V_T وكذلك الأضلاع المختارة. المصفوفة d[v] تعرض قيم الرؤوس في $V - V_T$ بعد أن يتم تحديثها.**

في الخوارزمية (3-6) جسم الحلقة while (الأسطر 10-13) ينفذ n-1 مرة. وعملية حساب $\min\{d[v]|v \in (V - V_T)\}$ (سطر 10) وحلقة for(أسطر 12-13) تنفذ في O(n) خطوة. ولذا فإن درجة التعقيد العامة لخوارزمية بريم هو $\Theta(n^2)$.

الخوارزمية(6-3): خوارزمية بريم التسلسلية لأصغر شجرة هيكلية.

```
1.   procedure PRIM_MST(V, E, w, r)
2.   begin
3.      V_T := {r};
4.      d[r] := 0;
5.      for all v ∈ (V - V_T ) do
6.         if edge (r, v) exists set d[v] := w(r, v);
7.         else set d[v] := ∞ ;
8.      while V_T ≠ V do
9.      begin
10.       find a vertex u such that d[u]:=min{d[v]|v ∈ (V - V_T )};
11.        V_T := V_T ∪ {u};
12.        for all v ∈ (V - V_T ) do
13.            d[v] := min{d[v], w(u, v)};
14.     endwhile
15.  end PRIM_MST
```

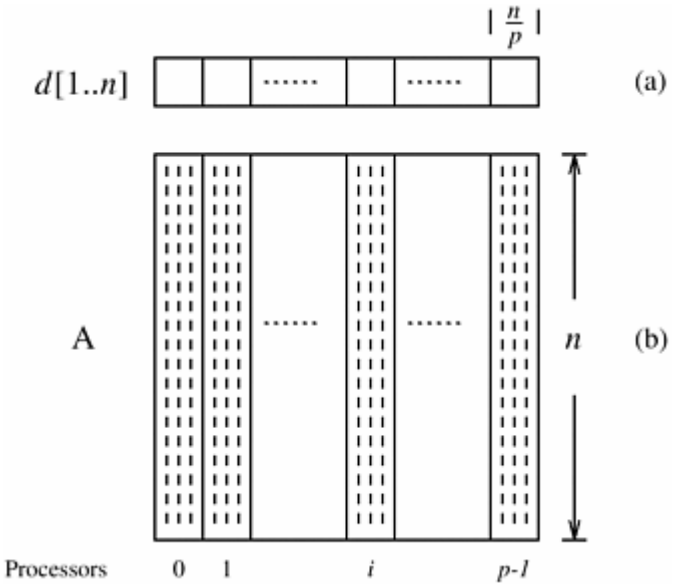
- **الصيغة المتوازية لخوارزمية بريم:**

تعتبر خوارزمية بريم تكرارية، وكل تكرار يضيف رأس جديد إلى أصغر شجرة هيكلية. ولأن قيمة d[v] للرأس v قد تتغير كل مرة يتم فيها إضافة رأس إلى V_T، فإنه من الصعب اختيار أكثر من رأس سويا لإضافتها إلى أصغر شجرة هيكلية. على سبيل المثال، في الشكل (3-23)، بعد أن يتم اختيار الرأس b، إذا تم اختيار الرأسين d و c سويا فإنه لا يمكن إيجاد أصغر شجرة هيكلية. بسبب أنه بعد اختيار الرأس d فقيمة d[c] يتم تحديثها من ٥ إلى ٢. وعلى ذلك فإنه ليس من السهل تنفيذ تكرارات مختلفة لعبارة While بالتوازي. ومع ذلك، فإنه يمكن تنفيذ كل تكرار بالتوازي كما يلي.

بفرض أن p هو عدد الإجرائيات، و n هي عدد الرؤوس في البيان. وتم تقسيم المجموعة V إلى p من المجموعات الجزئية. كل مجموعة فرعية لها n/p رأس متعاقب، والعمل المرافق لكل مجموعة فرعية يسند إلى إجرائيات مختلفة. بفرض أن V_i هي مجموعة فرعية من الرؤوس مسندة إلى الإجرائية P_i لكل i=0,1,...,p-1. وكل إجرائية P_i تخزن القسم من المصفوفة d الذي يطابق ل V_i *(بمعنى أن الإجرائية P_i تخزن d[v] بحيث أن $v \in V_i$)*. الشكل (3-24.a) يوضح هذا التقسيم. فكل إجرائية P_i تحسب أثناء كل تكرار لحلقة while التالي

$d_i[u]=\min\{di[v]|v \in (V - V_T) \cap Vi\}$.

وعند الحصول على القيمة الأصغر من جميع $d_i[u]$، وتخزن في الإجرائية P_0. والإجرائية P_0 تحتفظ الآن بالرأس الجديد u والذي سيضاف إلى V_T. وتقوم الإجرائية P_0 بإذاعة u إلى جميع الإجرائيات. كما أن الإجرائية P_i المسئولة عن الرأس u تحدد أن u ينتمي إلى المجموعة V_T، وفي النهاية، تقوم كل إجرائية بتحديث قيم d[v] للرؤوس الخاصة بـ(المحلية).



الشكل(3-24)**: تقسيم المصفوفة d ومصفوفة الجوار A إلى P إجرائية**

عند إضافة رأس جديد u إلى V_T، فإن قيم d[v] التي تحقق v ينتمي إلى ($V- V_T$) يجب أن يتم تحديثها. الإجرائية المطابقة لv يجب أن تعلم وزن الضلع (u,v). وبالتالي ستحتاج كـل إجرائية P_i إلى تخزين العمود لمصفوفة الجوار الموزونة المطابق للمجموعة V_i للرؤوس المـسندة إليها. المساحة المطلوبة لتخزين الأجزاء اللازمة من مـصفوفة الجـوار لكل إجـ ـرائية هي $\Theta (n^2/p)$. الشكل (3-24.b) يوضح التقسيم لمصفوفة الجوار الموزونة.

الفصل الرابع: البرمجة المتوازية

كما رأينا في فصلٍ سابق أن هناك تقنيات يمكن إتباعها لجعل خوارزمية أو مسألة معينة متوازية، ولكن يبقى لدينا الآن السؤال التالي: "كيف نقوم ببرمجة هذه المسألة الـمـزأة علـى الحاسب المتوازي؟"، والإجابة: يمكننا ذلك باستخدام نوع من البرمجة يطلق عليـه "البرمجـة المتوازية". إن التعبير عن التوازي في برامج المستخدم يتطلب تعـبيراً في اللغـات البرمجيـة وشكلها، فهي تتطلب مثلاً بعض التعليمات الأولية للتعبير عن التوازي بين مهمتين، وأخرى من أجل التخاطب والتزامن وما شابه ذلك.

البرمجـة المتوازيـة (Parallel Programming): هـي البرمجـة بلغـة تتـضمن البنـى أو الميـزات المتوازيـة.

ويمكن أن يتم بناء المميزات المتوازية من خلال بعض اللغات البرمجية التي تعتمـد مبـدأ التوازي في تصميمها مثل CSP(Communicating Sequential Process), OCCAM، أو يمكن توسيع لغات البرمجة التسلسلية من أجـل احتـواء تعليمـات التـوازي كلغـة Parallel-FORTRAN، Parallel-Pascal، Parallel-C. ويمكن كذلك إلحاق المزايا المتوازيـة إلى لغـة تسلسلية تقليدية مثل FORTRAN أو C/C++ وذلك باستخدام روتينات المكتبات مثل مكتبة MPI أو مكتبة PVM أو Pthreads(POSIX Threads).

وفيما يلي سوف نناقش بشيء من الإجمال لغة OCCAM ومن ثم سوف نعطي لمحة سريعة عن لغة FORTRAN-90 المتوازية، وأخيراً سنتعرض بشكل أوسع للبرمجة المتوازية باستخدام مكتبة MPI من خلال لغة C++.

4.1 لغة OCCAM

تعد لغة OCCAM إحدى اللغات المصممة خصيصاً للبرمجة المتوازية فهي تدعم التوازي الصريح والتوازي الظاهري على معالج واحد عن طريق تقسيم الوقت. وقد أخذ بعين الاعتبار عند تصميمها واختيار تعليماتا جميع خصائص العمل المتوازي. وقد تم تنفيذ OCCAM بعد دراسات طويلة جرت على البرامج واللغات المتوازية. واستفادت هذه اللغة من حصيلة الدراسات التي تمت على لغة CSP(Communicating Sequential Process) والتي تعد إحدى أوائل اللغات المصممة لهذا الغرض وأكثرها ملائمة لتوصيف وكتابة البرامج المتوازية، إلا أن لغة CSP لم تلق الشهرة نتيجة كونا لغة توصيفية، واقتصر استخدامها على مستوى مراكز الدراسات والمختبرات العلمية.

صممت لغة OCCAM لكي تنفذ بشكل خاص على معالجات الترانسبيوتر[1]، وهي تستثمر إلى حد كبير الخواص الصلبة والبنيوية لهذا المعالج. ولكون تعليمات هذه اللغة وخاصة الإجرائيات المرتبطة بالتفريع ذات مستوى قريب من المعالج فإن استخدامها اقتصر على المختصين في المعالجة المتوازية، ولم تلق رواجاً من قبل المستخدمين التقليديين.

يتألف البرنامج المتوازي وفق لغة OCCAM من عدد من المهمات تدعى Process وتنفذ كل منها على عقدة من عقد المعالجة المتوفرة. وتتألف كل مهمة من عدد من المهمات الجزئية المتسايرة (Concurrent Processes) التي تشارك في استثمار مصادر المعالج الوحيد.

[1] الترانسبيوتر Transputer: اختصار للجملة **trans**istor com**puter**. وهو حاسب آلي متكامل على شريحة واحدة، يتضمن ذاكرة وصول عشوائي(RAM) و وحدة حسابية للنقطة-العائمة (FPU)، ويصمم هذا الجهاز أساساً كوحدة بناء لأنظمة الحساب المتوازي.

وفي OCCAM يتم الاتصال بين المهمات المتسايرة عن طريق تمرير الرسائل من قنـاة إلى قناة.

ويوجد في OCCAM خمس مهمات أساسية، التخصيص، الإرسال، الاستقبال، تجـاوز، إيقاف، وفيما يلي إيضاح التركيب النحوي لها مع التمثيل:

التخصيص: هو إسناد قيمة تعبير حسابي إلى متغير

```
SYNTAX:      <variable>:=<expression>
Example:     x :=y +1
```

الاستقبال: استقبال قيمة من قناة. يستخدم الرمز "?" للتعبير عن الطلب

```
SYNTAX:      <channel>?<variable>
Example:     Ch ? x
```

الإرسال: إرسال قيمة تعبير إلى قناة. يستخدم الرمز "!" للتعبير عن النداء.

```
SYNTAX:      <channel>!<expression>
Example:     Ch !y +1
```

التجاوز: تجاوز المهمة الحالية

```
SYNTAX:      SKIP
Example:     SKIP
```

الإيقاف: إيقاف المهمة

```
SYNTAX:      STOP
Example:     STOP
```

وبشكل تدريجي (Hierarchical) تتألف أيضا كل مهمة جزئية من عدد مـن المهمـات الجزئية المتسايرة أو المتسلسلة، وقد يكون التوازي في هذه اللغة على مستوى التعليمة الواحدة أو على مستوى الإجرائية داخل المعالج الوحيد. وهكذا يتضمن البرنامج المتوازي عند تنفيذه على مئات المهمات المتسايرة والمتوازية.

يعبر عن التوازي في لغة OCCAM بتوزيع المهمات ذات المستوى الأعلى على معالجـات مختلفة، أما داخل المهمة فيعبر عنه باستخدام إجرائية التحكم PAR والتي تعني تنفيذ التعليمات أو الإجرائيات التي تلي على التوازي (توازي ظاهري).

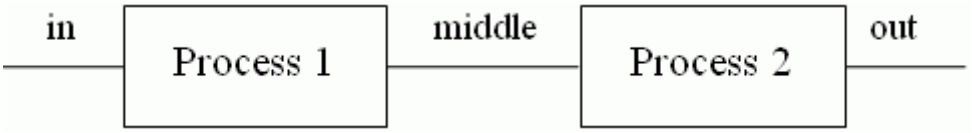
فللتعبير عن التوازي أثناء تنفيذ الإجرائيات S1,S2,S3 يمكن كتابة:

```
PAR        /*Ececute the following in parallel */
S1
S2
S3
```

كما يمكن تعريف الإجرائيتين S1,S2 مباشرة داخل إجرائية PAR على الشكل التالي:

```
CHAN OF INT in,out,middle;  /*Declaration of the channels */
PAR                         /*Execute in parallel*/
INT X:                      /*The first process*/
WHILE TRUE
SEQ                         /*Sequential execution of following*/
In ? X                      /*Read X from in channel*/
Middle ! X                  /*Write X on middle channel*/

INT X:                      /*The second process*/
WHILE TRUE
SEQ                         /*Sequential execution of following*/
MIDDLE ? X                  /*Read X from in channel*/
OUT ! X                     /*Write X on out channel*/
```

يمثل هذا البرنامج buffer ذو عنصرين وهو يتألف من مهمتين متسايرتين تتفاعلان عن طريق قناة الاتصال Middle كما يبين الشكل(1-4)



الشكل(1-4)

تستخدم تعليمة PAR أيضا كإجرائية تكرار فتقوم بخلق عدة مهمات متشابة ومتسايرة تنفذ على التوازي، ويتم ذلك على الشكل التالي:

```
PAR I=0 FOR N          /*create n process and execute then in parallel*/
S[I]
```

تمثل إجرائية PAR مهمة مؤلفة من عدة مهمات تنفذ على التوازي وتنتهي هذه المهمـة الأم عند انتهاء جميع المهمات الجزئية المؤلفة لها.

يستفاد من إجرائية PAR أيضا من أجل:

تعريف بنية البرنامج المتوازي على مستوى المهام العليا.

توضيع المهمات على المعالجات المختلفة.

وتأخذ هذه الإجرائية شكل PLACED PAR في هذه الحالة وتستخدم كما يلي:

```
PLACED PAR          /*Place in parallel*/
PROCESSOR 1         /*Processor naming*/
P1                  /*Task P1 on processor 1*/
PROCESSOR 2         /*Processor naming*/
P2                  /*Task P2 on processor 2*/
PROCESSOR 3         /*Processor naming*/
P3                  /*Task P3 on processor 3*/
```

كما لاحظنا في مثال سابق، تتوفر في لغة OCCAM إجرائية تحكم معاكسة لتعليمـة PAR، وهي تعليمة SEQ التي تعبر عن التنفيذ التسلسلي للتعليمات التي تليها.

يمكن أن يتم بناء المهمات باستخدام لغة OCCAM:

بشكل سكوني**:** حيث تتوضع في بداية التنفيذ مهمة على كل معالج.

بشكل آلي: حيث تقوم كل مهمة بخلق مهمات جزئية داخلية بالاستعانة بإجرائيـة PAR. ولكن لا يمكن لمهمة خلق مهمة جزئية على معالج آخر، فخلق المهمات محلي تدريجي.
تتفاعل المهمات فيما بينها عن طريق تبادل الرسائل عبر قنوات وحيدة الاتجاه تـصل بـين المهمتين. والتراسل في لغة OCCAM هو تراسل متزامن ويتم بعد الالتقاء بنقطة موعد فيمـا بين المهمتين المتراسلتين.
يمكن أن تتصل مهمتين من مهمات البرنامج المتوازي بعدة قنوات اتصال، وكل قناة تخصص لنقل نوع معين Type ومحدد من المعلومات، كما تنقل المعطيات وفق بروتوكـول معـرف ومحدد.
ويتم تحديد نوع المعطيات المتبادلة عند تعريف قناة الاتصال على الشكل التالي:

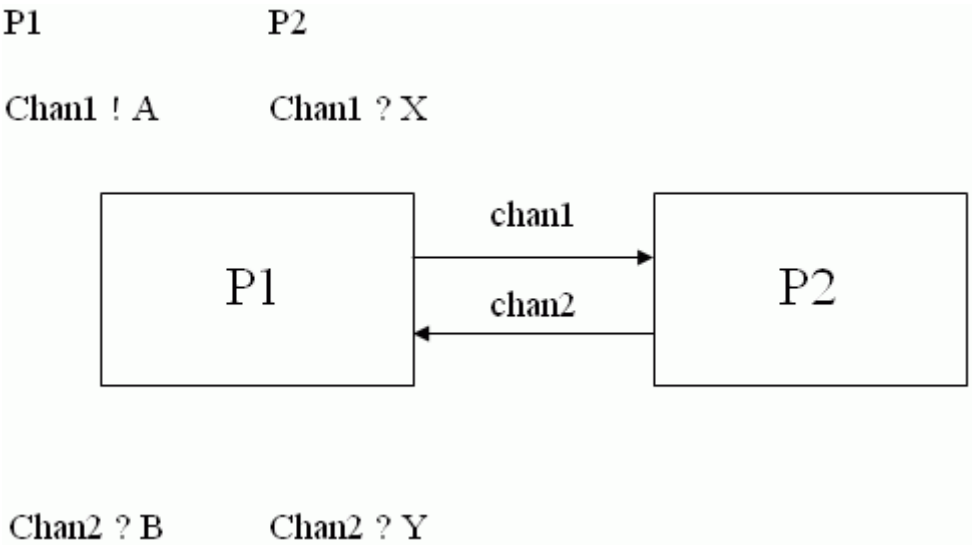
```
CHAN OF <TYPE>  <NAME1>,<NAME2>;
Example:
CHAN OF INT ch;              /*ch: channel of type integer*/
```

تعرف بروتوكولات نقل المعلومات في لغة OCCAM للتمكن من نقل معطيات مركبة ومن أنماط Types مختلفة عبر نفس القناة، فلإرسال عدد طبيعي يليـه عـدد صـحيح يعـرف البروتوكول التالي:

```
PROTOCOL int.real IS
INT;REAL32;
CHAN OF int.real ch:
INT inint;outint:
REAL32 inreal;outreal:
PAR
Ch ! outint;outreal
Ch ? inint;inreal
```

يتم تراسل المعطيات باستخدام تعليمتين : الأولى لإرسال المعلومات وهـي (!)، والأخـرى لاستقبال المعلومات وهي (؟). ويحدد نوع المعطيات المتبادلة كما ذكرنا عند تعريـف قنـاة الاتصال التي تصل بين مهمتين.

مثال: تتبادل المهمتين P1,P2 المعلومات باتجاهين عبر القناتين Chan1,Chan2 في الـشكل(4-2).



الشكل(2-4)

تتضمن لغة OCCAM أيضا إجرائية PRI لتحديد أولوية (Prioraty) تنفيذ مهمـة بالنـسبة للمهمات الأخرى فيكون لهذه المهمة أولوية التنفيذ بين المهمات المتسايرة على المعالج الواحد. وباستخدام إجرائية PRI، تنفذ المهمات وفقاً لترتيبها فالأولوية الأولى تكون للمهمـة الأولى ومن ثم للمهمة التي تليها وهكذا.

يوضح المثال التالي إجرائية PRI:

```
PRI PAR
P1          /*highest priority*/
PAR
P2          /*three process with middle priority*/
P3
P4
P5          /*lowest priority*/
```

تستخدم في لغة OCCAM أيضا إجرائيات التحكم المعروفة في اللغات البرمجية التقليدية مثـل FOR, WHILE,... وغيرها، واستخدامها يطابق استخدام ما يماثلها في اللغـات التقليديـة المعروفة.

وأخيراً تتضمن لغة OCCAM بالإضافة إلى إجرائية توضيع المهمات على المعالجات، إجرائيـة أخرى لتوضيع قنوات الاتصال التي تصل فيما بين المهمات المتوضعة على معالجات مختلفـة على القنوات الفيزيائية التي تصل فيما بين المعالجات.

ويوضح المثال التالي طريقة توضيع مهمة تتحكم بلوحة المفاتيح وتتصل عبر قنـاتي اتـصال بالمهمات الأخرى عبر قناتي اتصال على معالج Processor1. كما يوضح توضـيع قنـوات الاتصال المنطقية على قنوات الاتصال الفيزيائية:

```
PLACED PAR                  /*Plase in parallel*/
PROCESSOR 0                 /*Processor naming*/
  PLACE in AT link 0 in     /*Place channel in on link 0 in(physical channel)*/
  PLACE out AT link 1 out   /*Place channel out on link 1 out(physical channel)*/
  keyboard(in,out)          /*Place process keybord on Processor 0*/
PROCESSOR 1                 /*Processor naming Processor*/
  PLACE out 1 AT link 0 out /*Place channel out 1 on link 0 out*/
  PLACE in 1 AT link 0 in   /*Place channel in 1 on link 1 in*/
  Screen(in 1, out 1)       /*Place process screen on Processor 1*/
```

4.2 لغة FORTRAN- 90 :

هي لغة مشتقة من لغة FORTRAN ولكنها مصممة للحاسبات المتوازية التي من نمط SIMD حيث تقوم المعالجات المتزامنة بتنفيذ التعليمة نفسها في كل لحظة.
يظهر التوازي بالعمليات على المتجهات في هذه اللغة حيث يمكـن أن تكـون معـاملات العمليات الحسابية متجهات ذات بعد N فبدل كتابة الحلقة التالية:

```
for I=1,N
        A[I]=B[I] + C[I]
```

يمكننا باستخدام هذه اللغة كتابة تعليمات على المتجهات على الشكل التالي:

```
A(1:N)=B(1:N) + C(1:N)
T(1:N)=A(2:N+1)
 B(1:N)=2*T(1:N)
```

يقوم كل معالج بتنفيذ العملية المحددة على جزء المتجهة المخصصة له. ويتعاون المترجم ونظام التشغيل على هذه الحاسبات لتنفيذ هذه التعليمات المعقدة. كما يلعب التزامن دوراً أساسـياً في تسهيل تنفيذ التعليمة.
أما لغة HPF:Hight Performance FORTRAN فهي أحدث اللغات المتوازية المتوفرة علـى الحواسيب الحديثة، وهي تعتمد مبدأ توزيع المعطيات، وتعد إحدى أولى اللغات المعيارية لهذا النوع من الحواسيب.
تعتمد هذه اللغة مبادئ لغة FORTRAN-90 وتحتوي بالإضافة إلى ذلك تعليمـات لتوزيـع المعطيات على المعالجات المختلفة المتوفرة. أما عمليات الاتصال فيما بين المعالجات فهي شفافة بالنسبة للمستثمر.

4.3 واجهة تمرير الرسائل MPI *M*essage *P*assing *I*nterface

إن واجهة تمرير الرسائل MPI أنتجت في عام 1994(النسخة النهائيـة الأولى) لكي تكون مكتبة قياسية يعتمد عليها منتجو الحاسبات المتوازية بدلاً من أن ينتج كل موفر المكتبة الخاصة بـه كما كان هذا يحدث في السابق. فقد عرّفت هذه المكتبة معايير قياسية لتمرير الرسائل يمكن أن تستخدم لتطوير برامج تمرير الرسائل من خلال لغتي البرمجة C/C++ أو FORTRAN. وقد طورت مكتبة MPI بواسطة مجموعة من القطاعين الأكاديمي والصناعي، ولقيت دعماً واسعاً من قبل أغلب منتجي العتاد.

تحتوي مكتبة MPI على أكثر من 125 روتيناً، ولكن يمكن كتابة برامج متوازية كاملة الوظائف باستخدام 6 روتينات فقط وهي المعروضة في الجدول(4-1) هذه الروتينات الست تستخدم لبدء وإنهاء المكتبة، وتستخدم لإحضار معلومات عن بيئة التشغيل المتوازية التي يعمل عليها البرنامج بالإضافة إلى إرسال واستقبال الرسائل.

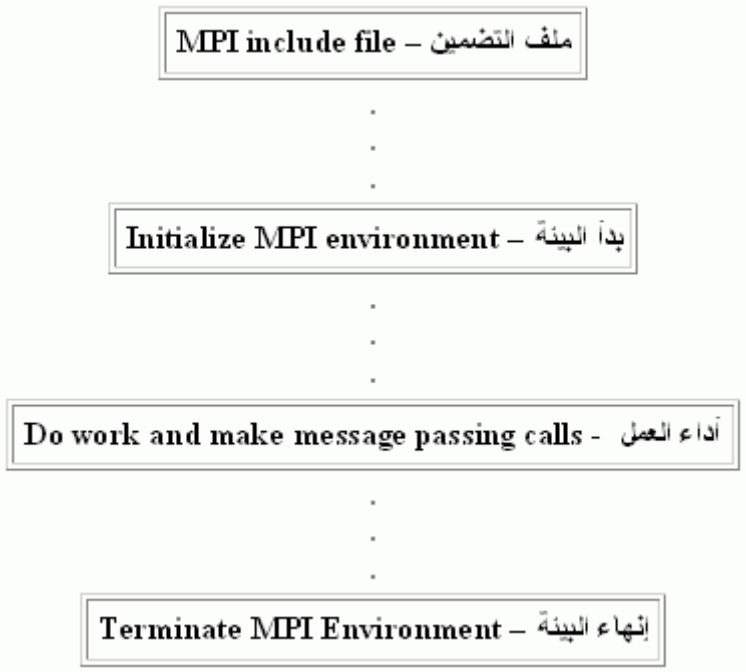
وفي هذا القسم سنتعرض لهذه الروتينات وشيء قليل من المفاهيم الأساسية والضرورية لكتابة برامج صحيحة وفعّالة لنموذج تمرير الرسائل باستخدام MPI.

الجدول(4-1): مجموعة من الروتينات الأساسية الخاصة بالمكتبة MPI

MPI_Init	بداية MPI
MPI_Finalize	إنهاء MPI
MPI_Comm_size	تحديد عدد المعالجات
MPI_Comm_rank	تحديد عنوان أو رتبة المعالج المستدعي
MPI_Send	إرسال رسالة
MPI_Recv	استقبال رسالة

4.3.1 الهيكل العام لبرامج MPI

يوضح الشكل الآتي الهيكل العام لبرامج MPI:



، MPI قبل أي استدعاء لأي روتين آخر في المكتبـة MPI_Init يتم استدعاء الروتين ووظيفة هذا الروتين هي تشغيل بيئة MPI. واستدعاء هذا الروتين أكثر من مرة خلال فتـرة عمل البرنامج سيتسبب في حدوث خطأ. ويستدعى الروتين MPI_Finalize لإنهاء الحساب. ويجب أن لا يستدعى أي روتين في MPI بعد استدعاء روتين الإنهاء.

ويجب أن يتم استدعاء الروتينين MPI_init و MPI_Finalize بواسطة كل الإجرائيـات و إلا سوف يكون سلوك البرنامج غير طبيعي. وفيما يلي طريقة الاستدعاء لهذين الروتينين في لغة C++:

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize()
```

إن جميع الروتينات و أنواع البيانات و الثوابت في MPI تُسْبَق بالأحرف MPI_ (على سبيل المثال MPI_Init). وهناك ثابت يتم إرجاعه عندما تتم العملية الحسابية بنجاح وهو MPI_SUCCESS. هذا الثابت وغيره من الثوابت وتراكيب البيانات في MPI كلها معرفة للغة C++ في الملف "mpi.h". ويجب أن يتم تضمين هذا الملف في جميع برامج MPI.

سنكتب الآن أول برنامج لنا باستخدام MPI وذلك بالاستفادة من الروتينين السابقين، وهو عبارة عن برنامج بسيط وفيه يقوم كل معالج لدينا بطباعة الرسالة الترحيبية "Welcome!" مع ملاحظة أنه في **برامج MPI يكون لدى كل معالج من معالجات الحاسب المتوازي نسخة خاصة به من نفس البرنامج**:

```
#include <iostream.h>
#include <mpi.h>  // لتضمين المكتبة في برنامجنا

int main(int argc, char ** argv)
{
     MPI_Init( &argc, &argv);
     cout << "Welcome!" << endl;
     MPI_Finalize();
}
```

نستخلص من البرنامج السابق عدة ملاحظات:

- قمنا بتضمين المكتبة mpi.h، وهذا يعطينا إمكانية التعامل مع جميع وظائف مكتبة MPI.
- لدينا في هذا البرنامج بداية ونهاية. كانت البداية حين استدعاء MPI_Init()، والتي تخبر نظام التشغيل بأن هذا البرنامج هو برنامج MPI وحينها سيقوم نظام التشغيل بعمل

الاعدادات اللازمة. أما النهاية فكانت حين استدعاء الروتين MPI_Finalize()، والتي تخبر نظام التشغيل بأن عليه تدمير أي شيء يتعلق في MPI.

- إذا كان البرنامج متوازي تماما -كما في برنامجنا هذا-، فإن العمليات التي تحدث بين عبارتي التهيئة والإنهاء لا تستخدِم أي اتصالات.

عندما نقوم بترجمة هذا البرنامج وتشغيله فسوف نحصل على مجموعة من رسائل "Welcome!" مطبوعةً على الشاشة، وعدد هذه الرسائل سيكون مساوياً لعدد المعالجات التي في الحاسب المتوازي الذي تم تشغيل البرنامج عليه، وتجدر الإشارة إلى أنه بسبب كون عملية طباعة الرسائل ستكون متزامنة، فإن الحاسب يجب أن يُسلسل الخرج بتتابع ليتم إخراجها على الشاشة.

4.3.2 المراسِلات (Communicators)

أحد الأشياء الرئيسية التي تستعمل في جميع برامج MPI الحقيقة هو ما يطلق عليه مجال الاتصال (communication domain). ومجال الاتصال هو مجموعة من الإجرائيات تسمح بحدوث اتصال فيما بينها. وبعض المعلومات حول مجال الاتصال تكون مخزنة في متغيرات من نوع MPI_Comm، والتي تدعى بالمراسلات. وهذه المراسلات تُستَخدم كبارامترات لجميع روتينات نقل الرسائل في MPI، لاحظ أن كل إجرائية يمكن أن تنتمي إلى مجالات تراسل مختلفة.

تستخدم المراسلات لتعريف مجموعة من الإجرائيات يمكن أن تتصل فيما بينها. وهذه الموعة من الإجرائيات تشكل مجال تراسل. وبشكل عام قد تحتاج جميع الإجرائيات للاتصال مع بعضها البعض، ولهذا السبب فإن MPI تُعرف مراسلات افتراضية تدعى

MPI_COMM_WORLD والتي تتضمن جميع الإجرائيات المستخدمة للتنفيذ المتوازي. ولكن في العديد من الحالات قد نحتاج أداء اتصال فقط ضمن مجموعة معينـة مـن الإجرائيـات. وباستعمال مراسلات مختلفة لكل مجموعة يمكن لنا أن نضمن أن الرسائل لا تتداخل أبدا مع رسائل مجموعة أخرى.

4.3.3 الحصول على معلومات عن بيئة التشغيل

إن الدالتين MPI_Comm_size و MPI_Comm_rank تُستخدَمان للحـصول علـى معلومات عن البيئة التي يعمل فيها البرنامج؛ فالأولى تستخدم لتحديـد عـدد الإجرائيـات والثانية لتحديد عنوان أو رتبة الإجرائية المستدعية، ويأخذ الروتينان الشكل:

```
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

إن الدالة MPI_Comm_size تُرجِعُ في المتغير Size عدد الإجرائيات التي تنتـسب لـال الاتصال comm.

كل إجرائية تتبع لال الاتصال تُعرّف بواسطة رتبتها rank. ورتبة الإجرائية هي عـدد صحيح يتراوح من صفر إلى حجم مجال الاتصال ناقص واحد. ويمكن معرفة رتبة الإجرائيـة باستخدام الدالة MPI_Comm_rank والتي تأخذ بارامترين: مجال الاتصال، و متغير صـحيح rank. المتغير rank يخزن رتبة الإجرائية. ويجب على الإجرائية التي تستدعي أي مـن هـذه الدوال أن تكون تابعةً لال الاتصال، و إلا سوف يحدث خطأ.

كنا قد كتبنا أول برنامج متوازي باستخدام دالتي البدء والإنهاء فقط، والآن سوف نقوم بتحسين البرنامج فبدلاً من أن يتم طباعة رسائل ترحيب فقط دون معرفة مصدرها، سـيتم الآن كتابة برنامج تقوم فيه كل إجرائية بطباعة رسالة ترحيبية وتوضح بجوارها أنـا هـي الإجرائية التي طبعت هذه الرسالة.

```
#include <iostream.h>
#include <mpi.h>

main(int argc, char *argv[])
{
      int npes, myrank;

      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &npes);
      MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

      cout <<"Welcome!  from process "<<myrank;
      cout <<"of "<<npes <<endl;
      MPI_Finalize();
}
```

عند تشغيل هذا البرنامج على حاسب لديه أربعة معالجات، فإن ناتج الطباعـة سـيكون شبيه للتالي:

```
Welcome!  from process 0 of 4
Welcome!  from process 2 of 4
Welcome!  from process 3 of 4
Welcome!  from process 1 of 4
```

لاحظ أن الناتج ربما لن يكون مرتبا بشكل صحيح وذلك لأن جميع المعالجـات تحـاول الطباعة على الشاشة بنفس الوقت، ونظام التشغيل هو الذي يقرر الترتيب.

4.3.4 تراسل البيانات في MPI

يمكن أن يتم إرسال الرسائل واستقبالها في MPI باستخدام الدالتين التاليتين: MPI_Send للإرسال و MPI_Recv للاستقبال. وفيما يلي سنوضح التركيب النحوي لكلا الدالتين، مع الشرح للبارامترات، ومثال للاستخدام.

التركيب النحوي لاستدعاء الدالة:

```
int MPI_Send(
      void*           message    /*in*/,
      int             count      /*in*/,
      MPI_Datatype    datatype   /*in*/,
      int             dest       /*in*/,
      int             tag        /*in*/,
      MPI_Comm        comm       /*in*/
      }
```

```
int MPI_Recv(
      void*           message    /*out*/,
      int             count      /*in*/,
      MPI_Datatype    datatype   /*in*/,
      int             source     /*in*/,
      int             tag        /*in*/,
      MPI_Comm        comm       /*in*/,
      MPI _Status*    status     /*out*/
      )
```

توصيف البارامترات:

- message - عنوان البداية لعازل (buffer) الإرسال والاستقبال.
- count - عدد العناصر في عازل الإرسال و الاستقبال.

- datatype - أنواع المعطيات للعناصر التي في عازل الإرسال.
- source - الرتبة (rank) للإجرائية لإرسال المعطيات.
- dest - الرتبة للإجرائية لاستقبال المعطيات.
- tag - لقب الرسالة.
- comm - يحدد مجال الاتصال.
- status - كائن الحالة.

ترسل الدالة MPI_Send المعطيات الموجودة في العازل (buffer) المشار إليها بواسـطة buf. يتألف هذا العازل من عناصر متتالية من النوع المحدد بواسطة البارامتر datatype. وعدد العناصر التي في العازل يحددها البارامتر count. وجهة الرسائل المرسلة بواسطة MPI_Send تحدد بواسطة البارامترين dest و comm. البارامتر dest هو رتبة (rank) للإجرائية الهـدف التي توجد في مجال الاتصال الذي يحدد بواسطة الراسل comm. وكل رسالة يرافقها لقـب tag وهو من نوع صحيح يستخدم للتمييز بين أنواع الرسائل المختلفة. يمكن أن يأخذ اللقب tag قيم تتراوح من صفر وحتى الحد الأعلى المعرف بواسطة MPI وهو MPI_TAG_UP.

أنواع البيانات في MPI:

إن أنواع البيانات في MPI التي تطابق تلك الأنواع الموجودة في لغـة C++ موضـحة في الجدول(2-4). فلكل نوع في سي هناك نوع مساوي له في MPI، هذا بالإضافة إلى أنـواع أخرى غير موجودة في لغة سي وهي MPI_BYTE و MPI_PACKED.

الجدول(2-4): يوضح أنواع البيانات في C++، ويوضح ما يقابلها في MPI.

أنواع البيانات في MPI	أنواع البيانات في C++
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	long double
MPI_BYTE	-----
MPI_PACKED	-----

أما الدالة MPI_Recv فتستقبل الرسائل المرسلة بواسطة إجرائية لها الرتبة التي يحملـها المتغير source وهذه الإجرائية موجودة في مجال الاتصال الذي يحدده البارامتر comm. يجب أن يكون لقب الرسالة المرسلة محدداً بواسطة البارامتر tag، فإذا كانَ هناك العديد من الرسائل لها نفس اللقب من نفس الإجرائية، فإنه يتم استقبال أي واحدة من هذه الرسائل. تـسمح MPI بتحديد رمز عام للبارامترات سواء بارامتر المصدر source أو اللقب tag، فإذا كـان المصدر source مهيئاً بالقيمة MPI_ANY_SOURCE فإن أي إجرائية في مجـال الاتـصال يمكن أن تكون المصدر للرسالة. وبنفس الطريقـة، إذا كـان اللقـب tag مهيـأ بالقيمـة MPI_ANY_TAG فإن الرسائل يتم قبولها جميعاً بأي لقب. يتم تخزين الرسالة المـستقبلة في المكان المشار إليه من قبل buf. يستخدم بارامتري count و datatype اللـذان في الـروتين MPI_Recv يستخدمان لتحديد طول العازل المهز. ويجب أن تكون الرسالة المستقبلة بطول مساوي أو أقل من هذا الطول. هذا يجعل المعالج المستقبل لا يعرف الحجم الدقيق للرسـالة

المرسلة. إذا كانت الرسالة المستقبلة أكبر من العازل المهز، فسينتج الخطـأ بتجـاوز الحـد المسموح، وسيعيد الروتين بالخطأ MPI_ERR_TRUNCATE.

بعد أن تستقبل الرسالة، فإنه يمكن استخدام المتغير status للحصول على معلومات حـول عملية الإرسال. وفي لغة C++ يتم تخزين متغير الحالة status في التركيب .MPI_Status وفيما يلي تطبيق لها كتركيب بثلاثة حقول:

```
typedef struct MPI_Status {
  int MPI_SOURCE;
  int MPI_TAG;
  int MPI_ERROR;
};
```

في المتغيرين MPI_SOURCE و MPI_TAG يتم تخزين المـصدر واللقـب للرسـالة المـستقبلة. وهمـا مفيـدان في حالـة اسـتخدام أي مـن MPI_ANY_SOURCE و MPI_ANY_TAG. وفي المتغير MPI_ERROR يتم تخزين رقم الخطأ للرسالة المستقبلة.

أيضا فإن بارامتر الحالة status يرجع بمعلومات عن طول الرسالة المستقبلة. ولكن هـذه المعلومات لا يمكن الحصول عليها مباشرة من المتغير status بل يمكـن أن نحـصل عليهـا باستدعاء الدالة MPI_Get_count والتي تأخذ الشكل التالي:

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype,
            int *count)
```

كما نشاهد فإن هذه الدالة تأخذ كبارامترات متغيري الحالة status ونـوع البيانـات datatype والتي حصلنا عليهما من الدالة MPI_Recv، أما الشيء الذي ترجعه لنـا الدالـة MPI_Get_count فهي تضع في المتغير count عدد العناصر.

وطريقة الاستخدام لعملية الإرسال والاستقبال يوضحها المثال:

```
int  mynode,  totalnodes;
int  datasize;      //عدد وحدات المعطيات التي سترسل أو تستقبل
int  sender;        //رقم الإجرائية المرسلة
int  receiver       //رقم الإجرائية المستقبلة
int  tag            // عدد صحيح يستخدم كلقب أو علامة للرسالة

MPI_Status status;        //متغير يحتوي معلومات عن الحالة

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
MPI_Comm_rank(MPI_COMM_WORLD, &mynode);

// datasize تحديد عدد وحدات المعطيات
double  *  databuffer =new double[datasize];

//Fill in sender,receiver,tag on sender/receiver processes,
//and fill in databuffer on the sender process.

if(mynode==sender)
      MPI_Send(databuffer,datasize,MPI_DOUBLE,receiver,
               tag,MPI_COMM_WORLD);


if(mynode==receiver)
      MPI_Recv(databuffer,datasize,MPI_DOUBLE,sender,tag,
               MPI_COMM_WORLD,&status);

// انتهت عملية الإرسال والاستقبال
```

4.3.5 برامج تطبيقية باستخدام MPI

بذا القدر نكون قد غطينا الجوانب الأساسية للبرمجة باستخدام MPI، وحـان الوقـت لكي نكتب برامج حقيقة، وفيما يلي سنختم هذا الفصل بعرضنا لعدة برامج، الهدف منـها هو تطبيق المعلومات التي وردت معنا سابقاً.

برنامج لإرسال واستقبال المعطيات

إن الهدف من هذا البرنامج هو توضيح وفهم عملية التراسل بين الإجرائيـات، وفيـه سوف نقوم بإنشاء مصفوفة على كل إجرائية، ولكن ستكون عملية التهيئة لكل المصفوفات في الإجرائية صفر(**P0**). وبعد أن تكون جميع المصفوفات قد هيئت في الإجرائية **P0**، تقـوم الإجرائية **P0** بالإرسال إلى كل إجرائية.

```
01 #include<iostream.h>
02 #include<mpi.h>
03
04 int main(int argc,char *argv[]){
05     int i;
06     int nitems =10;
07     int mynode,totalnodes;
08     MPI_Status status;
09
10     double *array;
11
12     MPI_Init(&argc,&argv);
13     MPI_Comm_size(MPI_COMM_WORLD,&totalnodes);
14     MPI_Comm_rank(MPI_COMM_WORLD,&mynode);
15
16     array =new double[nitems];
17
18     if(mynode ==0){
19           for(i=0;i<nitems;i++)
20                 array[i]=(double)i;
21     }
22
23     if(mynode==0)
24           for(i=1;i<totalnodes;i++)
25                 MPI_Send(array,nitems,MPI_DOUBLE,i,
                                 1,MPI_COMM_WORLD);
26     else
27           MPI_Recv(array,nitems,MPI_DOUBLE,
                 0,1,MPI_COMM_WORLD,&status);
28
29     for(i=0;i<nitems;i++){
30           cout <<"Processor "<<mynode;
31           cout <<":array["<<i <<"]="<<array[i]<<endl;
32     }
33
34     delete[]array;
35
```

```
36     MPI_Finalize();
37 }
```

تحليل البرنامج:

- قمنا في البداية بتهيئة MPI و جمع المعلومات (الأسطر 12 حتى 14).
- بعد ذلك قمنا بإنشاء مصفوفة على كل إجرائية باستخدام الحجز الديناميكي للذاكرة. (السطر 16)
- على الإجرائية P0 فقط(أي أن mynode=0)، قمنا بتهيئة المصفوفات لتحتوي على قيمة متغير الدليل (الأسطر من 18 حتى 21).
- على الإجرائية P0، نقوم باستدعاء روتين الإرسال MPI_Send بعدد (totalnodes-1). (الأسطر من 23 حتى 25)
- على كل الإجرائيات عدا الإجرائية P0 نقوم باستدعاء MPI_Recv وذلك لاستقبال الرسالة (الأسطر من 26 و 27).
- على كل إجرائية نقوم بطباعة ناتج عملية الإرسال والاستقبال (الأسطر من 29 حتى 32).
- على كل إجرائية، نقوم بتحرير الموضع الذاكرة الذي تحتله المصفوفة (السطر 34).

برنامج إرسال المعطيات ضمن حلقة (Ring)

وظيفة هذا البرنامج هي جمع أرقام المعالجات عن طريق تراسل المعالجات بأسلوب الحلقة ويتم تمرير أرقام المعالجات خلال الحلقة ليتم جمعها(عرفنا سابقاً أن كل معالج يُعطى رقم خاص به) فمثلاً لو كان لدينا أربعة معالجات فسيتم جمع القيم التالية ٠ + ١ + ٢ + ٣. في البرامج المتقدمة تكون البيانات المرسلة ذات قيمة وليس مجرد القيام بمثل هذه العملية.

```
#include "mpi.h"
#include <iostream.h>

/*Set up communication tags (these can be anything)*/
const int to_right=201;
```

```
const int to_left=102;

void main (int argc,char *argv[])
{
       int value,new_value,procnum,numprocs;
       int right,left;
       int sum,i;

       MPI_Status recv_status;

       /*Initialize MPI */
       MPI_Init(&argc,&argv);

       /*Find out this processor number */
       MPI_Comm_rank(MPI_COMM_WORLD,&procnum);
       /*Find out the number of processors */
       MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

       /*Compute number of the processor to the right */
       right =procnum +1;
       if (right ==numprocs) right =0;

       /*Compute number of the processor to the left */
       left =procnum -1;
       if (left ==-1)left =numprocs-1;

       sum =0;
       value =procnum;

       for(i =0;i <numprocs;i++){

              /*Send to the right */
              MPI_Send(&value,1,MPI_INT,right,to_right,
                           MPI_COMM_WORLD);

              /*Receive from the left */
              MPI_Recv(&new_value,1,MPI_INT,left,to_right,
                           MPI_COMM_WORLD,&recv_status);

              /*Sum the new value */
              sum =sum +new_value;

              /*Update the value to be passed */
              value =new_value;

              /*Print out the partial sums at each step */
              cout<<"PE "<<procnum<<": Partial sum ="<<sum<<endl;
       }

       /*Print out the final result */
       if (procnum ==0){
       cout<< "Sum of all processor numbers =" << sum << endl;
       }
       /*Shut down MPI */
       MPI_Finalize();
```

```
        return;
}
```

إذا شغل البرنامج السابق على جهاز بأربع معالجات، فسيكون له الخرج التالي:

```
PE 1: Partial sum =0
PE 2: Partial sum =1
PE 3: Partial sum =2
PE 0: Partial sum =3
PE 1: Partial sum =3
PE 2: Partial sum =1
PE 3: Partial sum =3
PE 0: Partial sum =5
PE 1: Partial sum =5
PE 2: Partial sum =4
PE 3: Partial sum =3
PE 0: Partial sum =6
PE 1: Partial sum =6
PE 2: Partial sum =6
PE 3: Partial sum =6
PE 0: Partial sum =6
Sum of all processor numbers =6
```

برنامج جمع سلسلة أعداد

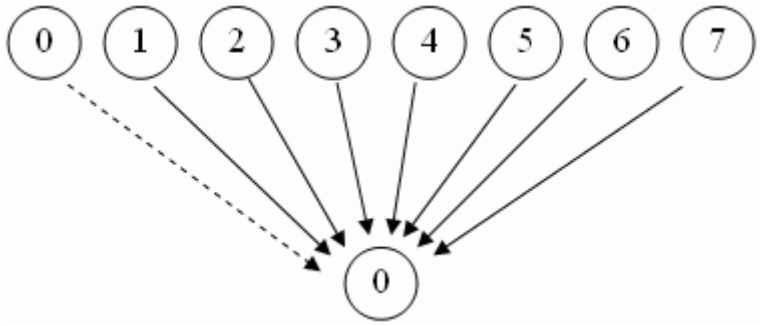
فيما يلي سوف نقوم ببرمجة مثال عددي بسيط، بحيث نريد أن نجمع كل الأعداد بداية من العدد ١ وحتى ١٠٠٠. في البداية سوف نرى التطبيق للكود التسلسلي:

```
#include<iostream.h>

int main(int argc,char **argv)
{
        int sum;

        sum =0;

        for(int i=1;i<=1000;i=i+1)
                sum  = sum +  i;
```

```
        cout  << "The sum from 1 to 1000 is:" << sum << endl;
}
```

الكود الموضح أعلاه يقوم بجمع الأعداد بالطريقة التسلسلية المعتادة والتي تعمل على معالج واحد، ولكن ماذا لو أردنا أن نؤدي هذا العمل على عدة معالجات؟ (مع انه ليس هنالك حاجة في مثل هذا العمل إلى عدة معالجات، وإنما المقصود في هذا المثال هو شرح كيفية تجزيء المسألة وبرمجتها).

لنفرض أن لدينا معالجين ونريد من المعالج الأول أن يقوم بجمع الأعداد من 1 وحتى 500، أما المعالج الثاني فسيقوم بجمع الأعداد من 501 وحتى 1000، وفي النهاية تجمع القيمتان الناتجتان معاً للحصول على الناتج الكلي للأعداد من 1 وحتى 1000. في الشكل (4-3) رسم تخطيطي عام، ولدينا فيه P معالج (في الشكل(4-3): P=8) والمسألة مجزئة إلى P مسألة فرعية، و عندما تنتهي جميع المعالجات تقوم بإرسال النواتج إلى المعالج P0 وذلك للتجميع النهائي للناتج.



الشكل(3-4): تتجمع كل المعلومات إلى معالج واحد باستخدام الإرسال والاستقبال

سنقوم بتجزيء العملية من خلال MPI:

ورد في السابق أن كل إجرائية يمكن لها أن تعرف العدد الكلـي للإجرائيـات الـتي ستُستَخدَم (عن طريق استخدام MPI_Comm_size)، وورد أيضا كيف للإجرائية أن تعرف نفسها أو رتبتها(باستخدام MPI_Comm_rank). ولنفرض أن المتغير mynode يخزن نـاتج استدعاء MPI_Comm_rank، ولنفرض أيضا أن المتغير totalnodes سيخزن ناتج اسـتدعاء MPI_Comm_size. وعلى هذا فالصيغة لتقسيم عملية الجمع عبر الإجرائيات معطاة بالكود التالي:

```
startval = 1000*mynode/totalnodes+1;
endval = 1000*(mynode+1)/totalnodes;
```

إذا كنا نستخدم معالج واحد فقط فستكون totalnodes =1 وسـتكون mynode =0، وإذن ستكون نقطة البداية startval=1 والنهاية endval =1000. أما إذا كنا نستخدم معالجين فستكون totalnodes =2 و ستكون mynode إمـا 0 أو 1. عنـد mynode =0 سـتكون startval =1 و endval =500، أما عند mynode =1 سـتكون startval =501 و endval =1000. وهكذا يمكن الاستمرار حتى يكون هنالك 1000 معالج، سيقوم عندها كل معـالج بأخذ عدد واحد (ليس هناك جمع!) وسترسل كل القيم إلى المعالج صفر(يبدأ ترقيم المعالجات من صفر) للتجميع.

عندما يكون لدينا قيمة بداية وقيمة نهاية لعملية الجمع فإن كل معالج يمكن أن ينفـذ عبارة التكرار for والتي تقوم بتجميع القيم التي بين startval و endval. وبعد أن ينتهي كل معالج من حساب المجموع الموكل إليه فإن جميع المعالجات (ماعدا المعالج صفر) ستقوم بإرسال المجموع الذي حصلت عليه إلى المعالج صفر، وبذا نحصل الناتج الكلي.

البرنامج التالي هو برنامج C++/MPI لإنجاز العملية السابقة:

```
#include<iostream.h>
#include<mpi.h>
```

```
int main(int argc,char **argv)
{
	int mynode,totalnodes;
	int sum,startval,endval,accum;
	MPI_Status status;

	MPI_Init(argc,argv);
	MPI_Comm_size(MPI_COMM_WORLD,&totalnodes);//get totalnodes
	MPI_Comm_rank(MPI_COMM_WORLD,&mynode); //get mynode

	sum =0;	//zero sum for accumulation
	startval =1000*mynode/totalnodes+1;
	endval =1000*(mynode+1)/totalnodes;

	for(int i=startval;i<=endval;i=i+1)
		sum =sum +I;

	if(mynode!=0)
		MPI_Send(&sum,1,MPI_INT,0,1,MPI_COMM_WORLD);

	else
	for(int j=1;j<totalnodes;j=j+1)
	{
		MPI_Recv(&accum,1,MPI_INT,j,1,MPI_COMM_WORLD,
			&status);
		sum =sum +accum;
	}
	if(mynode ==0)
		cout <<"The sum from 1 to 1000 is:"<<sum <<endl;
	MPI_Finalize();
}
```

برنامج الفرز الزوجي-الفردي

يقوم هذا البرنامج بفرز مجموعة أعداد بطريقة خوارزمية الفرز الزوجي-الفردي، وهي إحدى الخوارزميات التي شاهدناها في الفصل الثالث ضمن القسم "أمثلة للخوارزميات المتوازية". ولكن قبل كتابة البرنامج المتوازي سنكتب البرنامج التسلسلي للخوارزمية:

```
void ODD_EVEN(int n){

	int i,j,temp;

	for(i=1;i<=n;i=i+1)
	{
		if(i%2==1){
			for(j=0;j<=n/2-1;j=j+1)
				if(a[2*j]>a[2*j+1])
				{
					temp=a[2*j];
					a[2*j]=a[2*j+1];
					a[2*j+1]=temp;
				}
		}
		else{
			for(j=1;j<=n/2-1;j=j+1)
				if(a[2*j-1]>a[2*j])
				{
					temp=a[2*j-1];
					a[2*j-1]=a[2*j];
					a[2*j]=temp;
				}
		}
	}
}
```

نهاية البرنامج التسلسلي.

فيما يلي البرنامج المتوازي لنفس المسألة:

```
#include <stdlib.h>
#include <mpi.h> /* Include MPI's header file */

main(int argc, char *argv[])
{
  int n;         /* The total number of elements to be sorted */
  int npes;      /* The total number of processes */
  int myrank;    /* The rank of the calling process */
  int nlocal;    /* The local number of elements, and the array that stores
them */
  int *elmnts;   /* The array that stores the local elements */
  int *relmnts;  /* The array that stores the received elements */
  int oddrank;   /* The rank of the process during odd-phase communication */
  int evenrank;  /* The rank of the process during even-phase communication */
  int *wspace;   /* Working space during the compare-split operation */
  int i;
  MPI_Status status;
```

```
  /* Initialize MPI and get system information */
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &npes);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  n = atoi(argv[1]);
  nlocal = n/npes; /* Compute the number of elements to be stored locally. */

  /* Allocate memory for the various arrays */
  elmnts  = (int *)malloc(nlocal*sizeof(int));
  relmnts = (int *)malloc(nlocal*sizeof(int));
  wspace  = (int *)malloc(nlocal*sizeof(int));

  /* Fill-in the elmnts array with random elements */
  srandom(myrank);
  for (i=0; i<nlocal; i++)
    elmnts[i] = random();

  /* Sort the local elements using the built-in quicksort routine */
  qsort(elmnts, nlocal, sizeof(int), IncOrder);

  // Determine the rank of the processors that myrank needs to communicate
  //during the odd and even phases of the algorithm */
  if (myrank%2 == 0) {
    oddrank  = myrank-1;
    evenrank = myrank+1;
  }
  else {
    oddrank  = myrank+1;
    evenrank = myrank-1;
  }

  /* Set the ranks of the processors at the end of the linear */
  if (oddrank == -1 || oddrank == npes)
    oddrank = MPI_PROC_NULL;
  if (evenrank == -1 || evenrank == npes)
    evenrank = MPI_PROC_NULL;

  /* Get into the main loop of the odd-even sorting algorithm */
  for (i=0; i<npes-1; i++) {
    if (i%2 == 1) /* Odd phase */
      MPI_Sendrecv(elmnts, nlocal, MPI_INT, oddrank, 1, relmnts,
          nlocal, MPI_INT, oddrank, 1, MPI_COMM_WORLD, &status);
    else /* Even phase */
      MPI_Sendrecv(elmnts, nlocal, MPI_INT, evenrank, 1, relmnts,
          nlocal, MPI_INT, evenrank, 1, MPI_COMM_WORLD, &status);

    CompareSplit(nlocal, elmnts, relmnts, wspace,
                 myrank < status.MPI_SOURCE);
 }

  free(elmnts); free(relmnts); free(wspace);
  MPI_Finalize();
}

/* This is the CompareSplit function */
CompareSplit(int nlocal, int *elmnts, int *relmnts, int *wspace,
             int keepsmall)
{
  int i, j, k;
```

```
  for (i=0; i<nlocal; i++)
    wspace[i] = elmnts[i]; /* Copy the elmnts array into the wspace array */

  if (keepsmall) { /* Keep the nlocal smaller elements */
    for (i=j=k=0; k<nlocal; k++) {
      if (j == nlocal || (i < nlocal && wspace[i] < relmnts[j]))
        elmnts[k] = wspace[i++];
      else
        elmnts[k] = relmnts[j++];
    }
  }
  else { /* Keep the nlocal larger elements */
    for (i=k=nlocal-1, j=nlocal-1; k>=0; k--) {
      if (j == 0 || (i >= 0 && wspace[i] >= relmnts[j]))
        elmnts[k] = wspace[i--];
      else
        elmnts[k] = relmnts[j--];
    }
  }
}

/* The IncOrder function that is called by qsort is defined as follows */
int IncOrder(const void *e1, const void *e2)
{
  return (*((int *)e1) - *((int *)e2));
}
```


تم بحمد الله

أهم المراجع

[1] Daniel C.Hyde. Introduction to the Principles of Parallel Computation. Bucknell University, 1998

[2] Daniel C.Hyde. Introduction to the Programming Language Occam. Bucknell University, 1995

[3] Jim Buyens, Microsoft frontPage Version 2002 Inside Out, MSPress, 2001

[4] Karniadakis, G., and Kirby II, R., Parallel Scientific Computing in C ++and MPI. Cambridge University Press, 2003.

[5] Kumar, V , Gupta, A., Karypis, G., and Grama, A., Introduction to Parallel Computing, Second Edition. Addison Wesley Publishers, 2003.

[6] Message Passing Interface (MPI), http://www.mhpcc.edu/training/workshop/mpi/MAIN.html

[7] Microsoft Computer Dictionary, Fifth Edition. MSPress, 2002.

[8] Mikhail J. Atallah. Algorithms and Theory of Computation Handbook. CRC Press, 1998

[9] Null, L., and Lobur, J., The Essentials of Computer Organization and Architecture. Jones and Bartlett Publishers, 2003.

[10] Parhami, B., Introduction to Parallel Processing Algorithms and Architectures. KLUWER ACADEMIC PUBLISHERS, 2002

[11] Richard Jenkins. Supercomputers of today and tomorrow. TAB BOOKS Inc., 1986.

[12] TOP500 Supercomputer site, http://www.top500.org/.

[13] Tutorial on Message Passing Interface Programming MPI(MPICH for Windows)

[14] Message Passing Interface (MPI) Programming Lab http://www.dhpc.adelaide.edu.au/education/CS7933/lab/MPI/MPIlab.html

[15] **معجم مصطلحات الكمبيوتر، الدار العربية للعلوم، 2001.**

[16] **الربيعي وآخرون. المعجم الشامل لمصطلحات الحاسب الآلي والإنترنت. مكتبة العبيكان، 2001.**